SEEDEEP: A SYSTEM FOR EXPLORING AND QUERYING DEEP WEB DATA SOURCES

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of the Ohio State University

By

Fan Wang

Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
wangfa@cse.ohio-state.edu, BE/MS

Graduate Program in Computer Science and Engineering

The Ohio State University

2010

Dissertation Committee:

Dr. Gagan Agrawal, Advisor
Dr. Feng Qin
Dr. P. Sadayappan
ABSTRACT

A popular trend in data dissemination involves online data sources that are hidden behind query forms, thus forming what is referred to as the deep web. As compared to the surface web, where HTML pages are static and data is stored as document files, deep web data is stored in backend databases. Dynamic HTML pages are generated only after a user submits a query by filling an online form. Currently, hundreds of large, complex and in many cases, related and/or overlapping, deep web data sources have become available. The number of such data sources is still increasing rapidly every year.

The emergence of the deep web is posing many new challenges in data integration and query answering. First, unlike relational databases, where users have the direct access to the data tables in the databases, the metadata of the deep web databases (database schemas) and the complete set of data tuples stored in deep web databases are hidden from the data integration system. Second, most deep web data sources are created and maintained independently. Thus it is not uncommon for multiple data sources to have data redundancy and data overlap. Furthermore, similar data sources may provide data with different data quality and even conflicting data. Therefore, data source selection is of great importance for a data integration system. Third, deep web data sources in a domain often have inter-dependencies, i.e., the output from one data source may be the input of another data source. Thus, answering a query over a set of deep web data sources often involving accessing a sequence of inter-dependent
data sources in an intelligent order. Fourth, the common way of accessing data in deep web data sources is through standardized input interfaces. These interfaces, on one hand, provide a very simple query mechanism. On the other hand, these interfaces significantly constrain the types of queries that could be automatically executed. Finally, all deep web data sources are network based. Querying over such data sources often involve the use of various communication links. Both the data source servers and networking links are vulnerable to congestion and failures. Therefore, handling with fault tolerance issue is also necessary for a data integration system.

In our work, we propose SEEDEEP, an automatic System for Exploring and querying DEEP web data sources. The SEEDEEP system is able to integrate deep web data sources in a particular domain and provide search functionality on structured SQL queries, online aggregation queries and low selectivity queries for domain users. Currently, the SEEDEEP system is composed of five modules which includes schema mining, query planning, approximate query answering, query reuse and fault tolerance. The schema mining module can help to semi-automatically mine the metadata of deep web data sources and build data models for other modules in the system. The query planning module and the approximate query answering module are the core of the SEEDEEP system. The query planning module takes a structured query as input and generate a query plan over the set of integrated deep web data sources to answer the query based on a cost model. Currently, the query planning module is able to handle with Selection-Projection-Join queries, aggregation queries, and nested queries. For certain queries, it is hard to obtain the exact answer from the deep web within a reasonable period of time due to data access constraint specified on data source’s input interfaces or by the data source’s designer. These queries could be handled by the approximate query answering module in our SEEDEEP system. The
approximate query answering module is able to find approximate answers for online
aggregation and low selectivity queries using sampling in an effective and efficient
manner. The query reuse module explore the similarity between queries, and ac-
celerate the execution of a query by reusing previous query plans and cached query
data. Finally, the fault tolerance module deals with data source unavailability and
inaccessibility issues. During the execution of a query plan generated by the query
planning module, if some data sources become unavailable or inaccessible, the fault
tolerance module re-generate a partial query plan to replace the part of the original
query plan which becomes in accessible.

As part of our future work, we will extend our research in the following two di-
rections. First, we will look into the problem of understanding the deep web data, in
terms of data quality and data distribution. The knowledge of data quality and data
distribution is of great important to many querying problems. However, these infor-
mation cannot be easily obtained from the deep web due to limited accessibility of
the deep web data. As a result, we need to propose efficient and effective approxima-
tion methods to address the above problem. Second, we are going to look into query
planning problem for more advanced queries, such as correlated nested sub-queries,
existence queries (Is-there style queries), data mining queries and semantic queries.
Dedicated to my wife, Zixu Jiang, my mother, Yinxue Li, and my father, Huaijun Wang
VITA

1983 .................................. Born in Beijing, China
2005 .................................. B.E. in Computer Science and Engineering, China
2009 .................................. M.S. in Computer Science and Engineering, The Ohio State University
2006-Present .......................... Graduate Teaching/Research Associate, The Ohio State University

PUBLICATIONS


F. Wang and G. Agrawal, Extracting Output Metadata from Scientific Deep Web


**FIELDS OF STUDY**

Major Field: Computer Science and Engineering

Specialization: Database and System
# TABLE OF CONTENTS

Abstract .......................................................... ii  
Dedication ......................................................... v  
Vita ................................................................. vi  
List of Figures ................................................... xiii  

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
</table>
| 1       | Introduction ........................................... 1  
|         | 1.1 Motivation ........................................... 1  
|         | 1.2 Overview of SEEDEEP System ........................ 6  
|         | 1.3 Outline ............................................. 14  
| 2       | Related Work ............................................ 15  
|         | 2.1 Work Related with Answering Aggregation Queries Over Deep Web Data Sources ........................................... 15  
|         | 2.2 Work Related with Answering Low Selectivity Queries Over Deep Web Data Sources ........................................... 17  
|         | 2.3 Work Related with Query Answering Over Deep Web Data Sources ........................................... 19  
|         | 2.4 Work Related with Caching Mechanism for Query Answering Systems ........................................... 22  
|         | 2.5 Work Related with Supporting Fault Tolerance for Query Answering Systems ........................................... 24  
| 3       | Work Related with Mining Metadata of Deep Web Data Sources ........................................... 26  
| 3       | Effective and Efficient Sampling Methods for Deep Web Aggregation Queries ........................................... 28  
|         | 3.1 Motivation ........................................... 29  
|         | 3.2 Background ........................................... 34  
|         | 3.2.1 HDSampler and Its Estimators ..................... 34  
|         | 3.2.2 Adaptive Cluster Sampling ......................... 35  
|         | 3.3 New Sampling Algorithms ............................ 37  

viii
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.1</td>
<td>Problem Formulation</td>
<td>38</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Adaptive Neighborhood Sampling (ANS)</td>
<td>40</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Estimators for ANS Algorithm</td>
<td>45</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Sub-Space Based Two Phase Adaptive Sampling (TPS)</td>
<td>53</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Estimating MAX and MIN</td>
<td>58</td>
</tr>
<tr>
<td>3.4</td>
<td>Extension</td>
<td>58</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Multiple Data Sources</td>
<td>58</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Multiple Attributes of interest</td>
<td>59</td>
</tr>
<tr>
<td>3.5</td>
<td>Evaluation</td>
<td>60</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Data Sets</td>
<td>60</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Evaluation Metrics</td>
<td>61</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Performance of HDSampler on Synthetic Skewed Datasets</td>
<td>63</td>
</tr>
<tr>
<td>3.5.4</td>
<td>Evaluation of ANS Algorithm</td>
<td>63</td>
</tr>
<tr>
<td>3.5.5</td>
<td>Evaluation of TPS Algorithm</td>
<td>67</td>
</tr>
<tr>
<td>3.5.6</td>
<td>Comparison of Three Sampling Algorithms</td>
<td>68</td>
</tr>
<tr>
<td>3.6</td>
<td>Summary</td>
<td>79</td>
</tr>
<tr>
<td>4</td>
<td>Effective Stratification For Low Selectivity Queries Over Deep Web Data Sources</td>
<td>80</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>81</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Problem Formulation and Our Contributions</td>
<td>84</td>
</tr>
<tr>
<td>4.2</td>
<td>Background</td>
<td>86</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Stratification</td>
<td>86</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Harmony Search</td>
<td>88</td>
</tr>
<tr>
<td>4.3</td>
<td>Algorithm Overview</td>
<td>91</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Modeling Harmony Memory</td>
<td>91</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Harmony Search and Objective Function</td>
<td>94</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Sample Allocation</td>
<td>96</td>
</tr>
<tr>
<td>4.4</td>
<td>Bayesian Adaptation For Harmony Search</td>
<td>97</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Bayesian adaptation Overview</td>
<td>99</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Determining Probability Function $f_{\text{HMCR}}(\text{HMCR}</td>
<td>\theta)$</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Determining Prior Distribution and Bayesian Updates for $\theta$</td>
<td>101</td>
</tr>
<tr>
<td>4.5</td>
<td>Evaluation</td>
<td>102</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Data Sets</td>
<td>102</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Evaluation of Our Approach HarmonyAdp</td>
<td>104</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Comparison Between Adaptive and Non-Adaptive Versions</td>
<td>106</td>
</tr>
<tr>
<td>4.5.4</td>
<td>Comparison with Three Other Methods</td>
<td>107</td>
</tr>
<tr>
<td>4.6</td>
<td>Summary</td>
<td>113</td>
</tr>
<tr>
<td>5</td>
<td>Query Deep Web Data Sources: A Structured Keyword Query Approach</td>
<td>115</td>
</tr>
<tr>
<td>5.1</td>
<td>Motivation</td>
<td>115</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Motivating Examples</td>
<td>117</td>
</tr>
</tbody>
</table>
6.6 Summary ............................................. 184

7 Support Fault Tolerance for Deep Web Searchers: An Incremental Query
Plan Generation Approach ...................................... 185

7.1 Motivation ............................................. 186
7.2 Data Redundancy Model ..................................... 189
7.3 Problem Formulation ....................................... 191
  7.3.1 Incremental Query Plan Generation Problem .......... 191
  7.3.2 Problem Description ................................ 193
7.4 Incremental Query Planning Algorithm ......................... 194
  7.4.1 Minimal Impacted Subplan Generation ................. 195
  7.4.2 Maximal Fixable SubQuery Generation ............... 201
  7.4.3 Incremental Query Plan Generation ............... 207
7.5 Evaluation ............................................. 207
  7.5.1 Experiment Setup ................................... 207
  7.5.2 Experimental Results ................................ 209
7.6 Summary ............................................. 213

8 Extracting Output Metadata from Scientific Deep Web Data Sources . 214

8.1 Motivation ............................................. 214
  8.1.1 Related Problems and Existing Work .................... 217
  8.1.2 Our Approach .................................. 218
8.2 Problem Formulation and Solution Overview ................. 218
8.3 Sampling Model Approach .................................. 220
  8.3.1 Distribution Model ................................ 220
  8.3.2 Sampling Algorithm ................................ 222
  8.3.3 Sample Size Estimation .............................. 224
  8.3.4 Obtaining output schema attributes .................... 227
8.4 Mixture Model Approach .................................. 228
  8.4.1 Modeling of Data Sources ............................ 228
  8.4.2 Computing Data Source Borrowability ............... 230
  8.4.3 Validation Of Borrowed Attributes ................ 231
8.5 Evaluation ............................................. 231
  8.5.1 Label Classifier and Validation Algorithm Evaluation ... 232
  8.5.2 Evaluation Of Our Approaches ....................... 234
8.6 Summary ............................................. 238

9 Future Work ............................................. 239

  9.1 Understanding the Deep Web Data ......................... 239
  9.2 Querying the Deep Web Data ............................ 240

10 Conclusion ............................................. 242
Bibliography
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>The Plan of Answering Query Q1</td>
</tr>
<tr>
<td>1.2</td>
<td>System Infrastructure of SEEDEEP</td>
</tr>
<tr>
<td>3.1</td>
<td>$\frac{P(s_i</td>
</tr>
<tr>
<td>3.2</td>
<td>The Effect of $k$ and $\eta$ on AER - ANS Algorithm: (a) synskew3, (b) synskew7, (c) IBQ (skew=8)</td>
</tr>
<tr>
<td>3.3</td>
<td>The Effect of $k$ and $\eta$ on Sampling Cost - ANS Algorithm: (a) synskew3; (b) synskew7; (c) IBQ (skew=8)</td>
</tr>
<tr>
<td>3.4</td>
<td>ANS Performance w.r.t. Data Skew, Upper Limit Quantile=0.95</td>
</tr>
<tr>
<td>3.5</td>
<td>Analysis of Bias for ANS on IBQ Data, Cost Limit=200/500</td>
</tr>
<tr>
<td>3.6</td>
<td>ANS Algorithm With Post-stratification</td>
</tr>
<tr>
<td>3.7</td>
<td>The Effect of Selected Sub-space Size - TPS Algorithm: (a)synskew1; (b) synskew3; (c) IBQ (skew=8); (d) Auto (skew=0.7)</td>
</tr>
<tr>
<td>3.8</td>
<td>TPS Performance w.r.t. Data Skew</td>
</tr>
<tr>
<td>3.9</td>
<td>Compare AER for Three Algorithms on Synthetic Data</td>
</tr>
<tr>
<td>3.10</td>
<td>Comparing AER for Three Algorithms on IBQ (skew=8)</td>
</tr>
<tr>
<td>3.11</td>
<td>Comparing AER for Three Algorithms on Auto (skew=0.7)</td>
</tr>
<tr>
<td>3.12</td>
<td>ANS Performance Evaluation w.r.t. Neighborhood Proximity on synskew5</td>
</tr>
<tr>
<td>3.13</td>
<td>Sampling Cost Comparison for TPS and HDSampler Fixing AER Level on synskew1</td>
</tr>
<tr>
<td>3.14</td>
<td>Sampling Cost Comparison for TPS and HDSampler Fixing AER Level on Auto (skew=0.7)</td>
</tr>
</tbody>
</table>
8.1 Motivating Example 1: (a) Input Query Form for dbSNP, (b) Partial Output Page for rs12903896 from dbSNP, (c) Partial Output Page for rs7412 from dbSNP. 215

8.2 output schema attributes and Output Page Relationship 221

8.3 Performance Comparison for Different Approaches: (a) Performance Comparison on dbSNP; (b) Performance Comparison on KEGG; (c) Performance Comparison on JSNP; (d) Performance Comparison on Gene. 234
CHAPTER 1
INTRODUCTION

1.1 Motivation

A popular trend in data dissemination involves online data sources that are hidden behind query forms, thus forming what is referred to as the deep web [62]. As compared to the surface web, where HTML pages are static and data is stored as document files, deep web data is stored in backend databases. Dynamic HTML pages are generated only after a user submits a query by filling an online form. Currently, hundreds of large, complex and in many cases, related and/or overlapping, deep web data sources have become available. The number of such data sources is still increasing rapidly every year [8]. Thus, there is an increasing need for an automatic system that is able to manage and integrate heterogenous deep web data sources in a domain. We need to be able to facilitate exploration and queries on these deep web data sources. In this Ph.D. thesis, we are considering two types of query answering schemas. First, we consider answering complex and structured queries over multiple deep web data sources. Second, we consider the problem of approximate query answering for aggregation and low selectivity queries using sampling.

A specific motivating scenario of the first type of query answering schema, i.e., complex and structured queries, is as follows. In bioinformatics, Single Nucleotide
Polymorphisms (SNPs), seem particularly promising for explaining the genetic contribution to complex diseases [2, 114, 83]. Because over seven million Single Nucleotide Polymorphisms (SNPs) have been reported in public databases, it is desirable to develop methods of sifting through this information. Much information that biological researchers are interested in requires a search across multiple different web databases. No single database can provide all user requested information, and the output of some databases need to be the input for querying another database. Furthermore, for a query answering system, data integration and performance issues are also of great importance. The following motivating example further illustrates the need and challenges of building such a system.

Motivating Example 1 (Complex and Structured Queries): A biologist interested in SNP issues a query

\[
\text{SELECT g.SNPID, g.ORTH, g.HGNCID FROM GENE g.}
\]

The above query has the following intent: given a gene name \text{ERCC6}, we want to find the SNPIDs, the BLAST results and the corresponding HGNCID of gene \text{ERCC6}. To find the SNPIDs of \text{ERCC6}, we need to use gene name \text{ERCC6} as input to query on \text{dbSNP} data source to find SNPIDs. To find the BLAST information, we need to take the following three steps: 1) use \text{ERCC6} as input to query on \text{Gene} data source to obtain the proteins of \text{ERCC6} in human species and other orthologous species; 2) use \text{Protein} data source to find the sequences of the proteins obtained from step 1; 3) use the protein sequences from step 2 and the SNP information (amino acid position) obtained from \text{dbSNP} as input to do an alignment using \text{Entrez BLAST} data source. To find the HGNCID of \text{ERCC6}, we use \text{ERCC6} as input to query on \text{HGNC} data source. The plan of answering \text{Q1} is shown in Figure 1.1.

From this example, we can see that to answer such a query, the biologist must be able to complete the following steps:
Step 1: To learn the input and output schemas of biological deep web data sources so that she can manually identify appropriate data sources that could be used to answer the query;

Step 2: To understand the dependence relation among data sources so that she can manually figure out the order by which data sources should be accessed;

Step 3: To manually submit online queries to numerous query forms and keep track of the obtained results;

Step 4: To manually merge and organize the results from numerous data sources together in a systematic way.

The above procedure for answering one query is tedious and error-prone. For a biologist who may issue a number of such queries every day, a system that can automate this process will be highly desirable.

In designing and implementing such an automatic data integration and query answering system, we are facing many practical challenges. First, unlike relational databases, where users have the direct access to the data tables in the databases, the metadata of the deep web databases (database schemas) and the complete set of
data tuples stored in deep web databases are hidden from the data integration system. Second, most deep web data sources are created and maintained independently. Thus it is not uncommon for multiple data sources to have data redundancy and data overlapping. Furthermore, similar data sources may provide data with different data quality and even conflicting data. Therefore, data source selection is of great importance for a data integration system. Third, deep web data sources in a domain often have inter-dependencies, i.e., the output from one data source may be the input of another data source. Thus, answering a query over a set of deep web data sources often involving accessing a sequence of inter-dependent data sources in an intelligent order. Fourth, the common way of accessing data in deep web data sources is through standardized input interfaces. These interfaces, on one hand, provide a very simple interface. On the other hand, these interfaces significantly constrain the types of queries that could be automatically executed. Finally, all deep web data sources are network based. Querying over such data sources often involve the use of various communication links. Both the data source servers and networking links are vulnerable to congestion and failures. Therefore, handling with fault tolerance issue is also necessary for a data integration system.

Now, let us consider a specific scenario for the second type of query answering schema, which is approximate query answering for aggregation and low selectivity queries. In this thesis, for aggregation queries, we focus on the queries that require data enumeration. These are distinct from the aggregation queries that can be answered directly using a data source’s input interface. For example, “find the hotel with the lowest price in Boston on 2/15/2010” can be directly answered using a travel web-site, and specifying city and date. In contrast, the following aggregation query requires data enumeration:

Motivating Example 2 (Aggregation Queries): Consider a deep web data source
in the travel domain. Suppose a business student wants to study US aviation market, and for all major US airlines, he wants to obtain the average airfare from the US to Europe, across all flights in the next week.

To obtain the exact answer for this aggregation query, one needs to enumerate every pair of US and European city, and issue corresponding deep web queries. This can be extremely time-consuming, if not impossible. In general, finding the exact answer to an aggregation query that requires data enumeration is not practical due to the following reasons. First, it is extremely hard to obtain all possible input values to enumerate the data. Second, deep web data is returned over a network, and thus, executing a large number of queries can be extremely time consuming. Some data sources even charge access costs. Third, many deep web sources limit the number of queries a particular IP address can issue, or the number of data records can be returned, to protect their data from being completely downloaded, or to disallow a denial-of-service attack.

As stated earlier, users could only access deep web data through the input interface of data sources. Therefore, some attributes, in which users may be interested, are not directly queriable through the input interface of deep web data sources. This imposes a great challenge for handling low selectivity queries whose selective attribute is not directly queriable through the deep web data source input interface. Let us consider the following example.

**Motivating Example 3 (Low Selectivity Queries):** Biologists want to obtain the average frequency of the SNPs which are related with a certain type of rare breast cancer. In this example, the target attribute is *SNP frequency* and the selective attribute is *SNP function*. SeattleSNP\(^1\) is a widely used deep web data source

\(^1\)http://pga.gs.washington.edu/
for searching SNP related information. However, SNP Function, which is the selective attribute of the above query, is not directly queriable on the input interface of SeattleSNP. Therefore, to find exact answers to the above query, we need to enumerate every SNP from SeattleSNP and filter out the SNPs whose function is not related with the specified type of breast cancer. Due to the reasons that we mentioned for aggregation queries above, this procedure could be very time-consuming, if not impossible. Sampling is a widely used method to find approximate answers for low selectivity queries. However, a naive random sampling strategy may not work well because the low selectivity of the query could filter out all the data records which are sampled, yielding poor estimation results.

1.2 Overview of SEEDEEP System

In our work, we propose SEEDEEP, an automatic System for Exploring and quEry-ing DEEP web data sources. Distinct from existing deep web mining systems in the e-commerce domain [64, 27, 26, 65], which mainly focus on schema matching, the SEEDEEP system is able to integrate deep web data sources in a particular domain and provide keyword based search feature for domain users. Currently, the SEEDEEP system is composed of five modules which includes schema mining, query planning, approximate query answering, query reuse and fault tolerance. The system infrastructure of SEEDEEP is shown in Figure 1.2. We briefly introduce each module in SEEDEEP as follows.

Schema Mining Module

The popularity of deep web for data dissemination is leading to new problems in data integration. Particularly, to enable data integration from multiple deep web data sources, one needs to obtain the metadata for each of the data sources. Obtaining the
metadata, particularly, the output schema, can be very challenging. This is because, given an input query, many deep web data sources only return a subset of the output schema attributes, i.e., the ones that have a non-NULL value for the corresponding input. The schema mining module is used to automatically mine the metadata deep web data sources and build data models for other three modules in the SEEDEEP system.

The schema mining module uses two approaches which are sampling model approach and mixture model approach to automatically mine the output metadata of deep web data sources. The sampling model approach is based on a distribution model of deep web data source output attributes. We found that a modest sized sample of output pages could discover output attributes with relatively high recall. Based on this idea, we proposed a sampling based algorithm to discover output attributes based on a simple random input samples. The size of the sample is estimated by
a sample size estimator which could bound the sampling error within a confidence interval. The mixture model approach is based on an observation that there is likely some redundancy across data sources [15]. Thus, attributes could be shared among different data sources. We model a data source as a probabilistic data source model that generates output attributes with certain probabilities. Since an attribute could be shared by multiple data sources, we consider the probability of an attribute generated by a data source as being determined by a mixture model composed of the probabilistic data sources of similar data sources.

**Query Planning Module**

Most deep web data sources only support simple text interfaces for querying them. These interfaces are easy to use, but their expressive power is quite limited. Therefore, processing complex structured queries over deep web data sources usually involves a great amount of manual work. At the same time, such functionality is becoming increasingly desirable and even crucial in many areas.

Recently, two ways to deal with deep web data have been articulated, which are, *surfacing* and *virtual integration* [98]. An intuitive way of supporting complex structured queries over deep web data is to *surface* the data and store it in a relational database. Then, standard query languages and techniques can be used. However, data in many deep web data sources is continually updated. Thus, the surfacing approach would lead to a data consistency problem.

In the *virtual integration* approach, a mediated schema for the data sources in a domain can be created by schema matching techniques [60]. Then, users can pose queries over the *mediated schema* that is exposed to them as a web form. The mediator approach has been very well studied, prior to the interest in the deep web. Some of the very well known examples of mediator systems that have been built are SIMS [5],
Information Manifold [88], TSIMMIS [51], and MedMaker [104]. A number of recent efforts on building deep web querying systems can also be viewed as mediator systems. Among these, the systems described in [17, 118] mainly deal with optimizing the execution of queries on the web, but do not consider query planning for queries expressed in a high-level, highly-expressive query language. The systems proposed in [84, 127, 118] have query planning algorithms for answering queries over the deep web, but they cannot support aggregation or nested queries.

Our query planning module is capable of supporting complex structured queries on the set of integrated deep web data sources in the SEEDEEP system within a domain. We develop and evaluate a framework for querying a set of integrated deep web data sources. Currently, the query planning module could support three types of advanced query: \textit{SPJ queries}, \textit{aggregation queries}, and \textit{nested queries}.

Given the metadata extracted using the schema mining module, a data source inter-dependency directed graph, which captures the dependency relation between correlated data sources, are built as our query data model. An example of the dependency graph model is shown in Figure 1.2. Our approach for supporting advanced structured queries is as follows. 1). If the query contains nested queries, we divide the original query into a list of \textit{simple queries}, i.e., the queries without nested queries. Then, for each \textit{simple query}, we extract the search terms and necessary predicates to understand the query type. 2). We use our novel query planning algorithms to generate a query plan for each simple query. 3). The query plans for all simple queries in the original query are \textit{combined} and \textit{merged} together to form the final query plan. Finally, several optimization techniques that we have developed are applied to speed up the execution.
Approximate Query Answering

The approximate query answering module is capable of finding approximate answers for aggregation queries which require data enumeration and low selectivity queries in presence of limited data access over the deep web by using intelligently designed novel sampling methods.

As described above, deep web aggregation queries requiring data enumeration can only be answered approximately, using sampling. We expect that users will be satisfied with a reasonably accurate, but approximate, answer. This, however, requires effective and efficient sampling. Even generating random samples from hidden databases can be challenging. Recently, Dasgupta et al have addressed this problem by developing HDSampler [41, 39], which is able to select a *simple random sample* (SRS) from hidden databases.

The random sampling approach like HDSampler may not work effectively on many deep web data sources because it has low estimation accuracy on skewed data and has high sampling cost. In this thesis, we proposed new techniques to support approximate aggregation queries over deep web data sources that may have skewed data. We have developed and evaluated two new *adaptive sampling* algorithms. They are the *Adaptive Neighborhood Sampling* (ANS) algorithm and the *sub-space based Two Phase adaptive Sampling* (TPS) algorithm. Both these algorithms have two advantages over HDSampler. First, both of them can provide accurate estimation for aggregation queries on skewed data, without requiring any knowledge of the hidden data distribution or other statistical information. Second, for data with a low skew, the TPS algorithm incurs significantly lower sampling costs compared to HDSampler. Without any prior knowledge of the data distribution, our algorithms yield accurate estimation on skewed data. In addition, the TPS algorithm has lower sampling costs. Our experiments show that for data with a moderate or a large skew, ANS and TPS
yield more accurate estimates, outperforming HDSampler by a factor of 4 on the average. For data with a small skew, TPS method only incurs one-third of the sampling cost of HDSampler.

For low selectivity queries, like the ones illustrated in Motivating Example 3, stratification is usually used to find more accurate answers. Specifically, we partition the entire data set into multiple sub-groups, called strata, such that the values of the selective attribute(s) of the query become relatively homogenous within each stratum. This objective is best achieved when the variability of the partitioning attribute(s), in our case the selective attribute(s), within each stratum is minimized and between different strata is maximized. After stratification, we know that the query selection predicate will favor certain strata. Then, those favored strata can be heavily sampled so as to obtain a better estimation result for the query. In detail, suppose a total number of \( n \) samples are drawn from the data set, the number of samples selected from each stratum can be determined by existing sampling allocation methods, such as Neyman allocation \([111]\) and Bayes-Neyman allocation \([78]\).

Therefore, the critical problem is to find an ”optimal stratification”, in other words, to find multiple breaking points of the partitioning attribute on which the stratification is performed. In the literature, methods have been proposed for this purpose, such as clustering based stratification \([111]\) and outlier indexing \([28]\). The above methods require the knowledge of either the distribution or summary statistics of the partitioning attribute, which can be easily obtained in the context of relational databases. However, in the deep web scenario, if the partitioning attribute is not in the input query form, it is not directly queriable, and as a result, the above methods cannot be applied. In this thesis, we developed a Bayesian adaptive harmony search stratification algorithm for a non-queriable selective attribute based on an auxiliary attribute. In our proposed approach, we take certain directly queriable attribute on
the input query form, the *input attribute r:Input*, as the *auxiliary attribute*. Then, we develop an alternative to the recently proposed meta-heuristic algorithm, harmony search algorithm [91], that we call Bayesian adaptive harmony search algorithm to find robust stratification plans for the non-queriable selective attribute based on the auxiliary attribute. The stratification thus obtained can accurately reflects the distribution of the *hidden selective attribute*. We have experimentally show that the estimation accuracy obtained based on the stratification from our method outperforms three existing method at least by a factor of 5. Furthermore, The estimation accuracy we obtained is consistently higher than 95% even for extreme low selectivity query (0.01% selectivity).

**Query Reuse Module**

Executing a query plan often involves accessing a sequence of inter-dependent deep web data sources as illustrated in Figure 1.1. Queries over such setting can be extremely time consuming due to the load on the server-side and the network delays. The query reuse module provides an effective and novel query caching mechanism which greatly accelerates the execution of a query plan taking advantage of cached data.

We propose a *query-plan-driven* query caching approach which not only caches the previously extracted data, but more importantly, caches the query plans for the queries. Thus, we generate a query plan for the new query by reusing the cached query plans of closely related previous queries, which increases the *possibility of data reuse*. An effective data reuse could convert a remote data source access to a local disk access, which greatly reduces the execution time of a query plan. Specifically, given a new query $NQ$, the query reuse module first finds a list of previous queries, denoted as $PQs = \{PQ_1, \ldots, PQ_n\}$, which are similar to $NQ$ based on a similarity metric.
Second, from each $PQ_i$, a $\Psi$ selection algorithm selects a query sub-plan $SubP_i$ such that among all valid sub-plans of $PQ_i$, $SubP_i$ maximally covers $NQ$ and has the smallest size. The query plan for $NQ$ is generated using the list of $SubP_i$ based on a modified query planning algorithm implemented in the query planning module. When generating the query plan for the new query $NQ$, the modified planning algorithm gives detours to the reusable previous sub-plans selected by the $\Psi$ algorithm so that the previously cached data can be reused.

**Fault Tolerance Module**

Executing a query plan often requires data access over wide-area networks, thus involving a large number of remote data sources and use of various communication links. Both the servers and networking links are vulnerable to congestion and failures. This can lead to an unpredictable unavailability or inaccessibility, which can disrupt access to the information.

we are proposing an approach which can be considered as data redundancy based incremental query processing. During the processing of the original query plan, if some data sources become unavailable, the simplest action is to completely discard the current execution, and generate a new plan for the original query. The obvious disadvantage of this approach is that it wastes a significant amount of work has already been performed. In our approach, the query processing will not be terminated. Instead, the execution of the part of the query plan which becomes unusable due to the unavailable data sources will be suspended. We dynamically adapt query processing by exploiting the data redundancy that is found across deep web data sources, i.e., many deep web data sources in a particular domain could overlap in terms of data content. We incrementally generate a partial new query plan by bringing in new data
sources that were not in the original query plan to replace the subplan that became inaccessible.

1.3 Outline

The rest of the thesis is organized as follows. In Chapter 2, we compare our work with existing work on related topics. In Chapter 3, we propose two novel sampling algorithms to intelligently select samples from the deep web data sources so that deep web aggregation queries requiring data enumeration can be answered accurately and efficiently. In Chapter 4, we describe a Bayesian adaptive harmony search algorithm to effectively answer aggregation queries with low selectivity in presence of data access limitation. Our algorithm could find an optimal stratification on the limited accessible selective attribute based on auxiliary attribute, and a stratified sampling algorithm could be applied to find accurate estimates for the query. In Chapter 5, we describe the query planning module of SEEDEEP in detail. We introduce the data model used for query planning algorithms. Three query planning algorithms are presented for three different types of keyword query we are considering. In Chapter 6, we give the detailed design of the query-plan-driven query caching strategy which is implemented in the query reuse module of our system. The fault tolerance module and the dynamic query plan adaptive algorithm is introduced in Chapter 7. In Chapter 8, we present the sampling model method and the mixture model method for mining metadata of deep web data sources. Some future work is described in Chapter 9 and we conclude in Chapter 10.
CHAPTER 2
RELATED WORK

In this chapter, we compare our work with existing work on topics related with the four modules in our SEEDEEP system.

2.1 Work Related with Answering Aggregation Queries Over Deep Web Data Sources

We now compare our work about answering aggregation queries over deep web data sources with existing work on related topics, including online aggregation, approximate query answering, and hidden web sampling.

Online Aggregation: Hellerstein et al [66] have developed techniques for online aggregation. Approximate answers for aggregation queries are generated and further refined when all data has been processed. To make online aggregation scalable to larger data sizes, Jermaine et al [75] proposed a scalable online aggregation method within the DBO engine. We are considering a distinct problem. First, we focus on sampling algorithms to obtain a sample with a low sampling cost and developing efficient estimators to provide accurate estimates. Second, in our case, query is answered only using the sample, but in online aggregation, eventually, all data is processed and an exact answer is obtained. Finally, skewed data is not considered in the above online aggregation efforts.
Approximate Query Answering: To provide better estimates for aggregation queries on skewed data, efforts have focused on pre-processing the database. Histograms [106] and wavelets [24] are pre-computed and used to provide more accurate answers for aggregation or other queries. Chaudhuri et al [30, 7, 28, 29] have conducted extensive studies on approximate aggregation queries answering using workload information and biased samples. The approaches include partitioning the database into a list of fundamental regions [29, 30] or groups [7], based on a given workload of queries. Each fundamental region or group is considered as a stratum. Then, a stratified random sample is selected from each stratum based on some property of the stratum. The sample thus chosen is used to answer future queries, which are expected to be similar to the workload initially provided. To handle skewed data, an outlier indexing technique has been developed [28]. Joshi et al designed sampling methods and estimators for low selectivity queries [78] and subset-based queries [79]. Wu et al [131] have developed a Bayesian method for estimating the extreme values in a dataset based on the learned characteristics of a previous workload. Palmer et al [103] have used density biased sampling to improve data mining queries on a large dataset. Jermaine et al [74, 77] proposed the APA (approximate pre-aggregation) and APA+ methods to answer aggregation queries on skewed data using an SRS combined with a small set of statistics about the data.

The above approaches cannot be applied in the deep web scenario. First, to build fundamental regions, stratums, histograms, wavelets, or gathering supplemental data statistics, at least one scan of the entire database is required. Without the access to the full data, as is the case in the deep web, this is not possible. Another distinct aspect of our approach is that we do not assume that we have a workload and future queries are going to be similar to the queries in the workload.

Hidden Web Sampling: The prominent work in this area is by Dasgupta et al. [41,
39], who have developed HDSampler. We have extensively compared our work with their method. Afrati et al [1] have developed an adaptive sampling algorithm for answering aggregation queries on web-sites with hierarchical structures. They assume that a hierarchical structure partitions a dataset into groups, and focus on adaptively adjusting the sample size assigned to each group based on the estimation error in each group. In our problem, no such hierarchical structure assumed. We focus on adaptively selecting samples, not adjusting sample sizes.

2.2 Work Related with Answering Low Selectivity Queries Over Deep Web Data Sources

We now compare our work with existing work on related topics, including optimal stratification, answering queries over the deep web, answering low selectivity queries, and adaptive harmony search.

**Optimal Stratification:** The method of choosing the best boundaries that make strata internally homogenous is known as optimum stratification. Ideally, stratification should be done using the survey variable $y$. However, in many cases, $y$ is unknown, and as a result, many stratification methods have been proposed to perform stratification based on an auxiliary variable $x$. The well-known cumulative root frequency method of stratum construction (D&H method), derived by Dalenius and Hodges [36, 37], depends critically on the assumption that the distribution of $x$ in each stratum is approximate uniform. To further simplify the computation of the D&H method, Ekman [49] and Gunning and Horgan [57] have both proposed several other approximation methods. These methods are based on the Dalenius and Hodges’s assumption, and even add more assumptions on the auxiliary variable $x$. As we have experimentally shown and analyzed in Chapter 4, these existing methods
are not appropriate for our problem because of the following two reasons. First, when the correlation between the \( x \) and \( y \) is weak, the assumptions in these methods do not hold true. Second, these methods are not designed for low selectivity queries. There are stratification methods proposed for data following certain distribution, such as uniform and right triangular distributions [85], exponential [86] and triangular distribution [87]. However, in our problem, we need to handle dataset with an arbitrary and/or unknown distribution.

**Answering Database (Low Selectivity) Queries Using Sampling:** Answering database queries by sampling has been extensively studied in the literature [67, 102]. Chaudhuri et al [30, 28, 29] have conducted extensive studies on approximate aggregation queries answering using stratification. The approaches include partitioning the database into fundamental regions [29, 30] and generating outlier indexing [28]. The above approaches cannot be applied in the deep web scenario, as they need to know critical statistics or distribution about dataset. With support for only limited data accesses, as is the case on the deep web, this is not possible. Joshi et al designed sampling methods and estimators for low selectivity queries [78] using Bayesian-Neyman Allocation. The key difference between our work and their approach is that they only focus on allocating samples into stratum, whereas we also consider determining optimum stratification.

**Answering Queries Over the Deep Web:** The prominent work on deep web sampling and query answering includes online aggregation [66, 75] and hidden web sampler developed by Dasgupta et al. [41, 39]. In online aggregation, approximate answers for aggregation queries are generated and further refined when all data has been processed. Jermaine et al [75] proposed a scalable online aggregation method within the DBO engine. The hidden web sampler focuses on generating true random samples from the deep web. Recently, Dasgupta et al [40] developed a unbiased
estimator for size and other aggregates over hidden web data sources. Their approach could use a small number of queries to produce unbiased estimates with a small variance. Joshi et al designed sampling methods and estimators for subset-based queries [79]. The key distinct aspect of our work is that we focus on handling low selectivity aggregation queries.

**Adaptive Harmony Search:** Some work has been done in providing adaptive features on harmony search. Mahdavi et al [100] proposed a linear adaptive method to only update the PAR parameters in harmony search. The value of the PAR parameter is changed in a pre-defined linear pattern during the execution of the algorithm. Similarly, Hasancebi et al [59] adapt the two harmony search parameters according to a pre-defined logistic normal distribution. Unlike the existing work, our Bayesian adaptive approach could modify the parameters based on the previous status of the harmony memory. This makes our method more robust.

### 2.3 Work Related with Query Answering Over Deep Web Data Sources

We now compare our work about query answering over deep web data sources with existing work on related topics, including query mediation systems, query processing and optimization on web data sources, keyword search on relational databases, and deep web crawling systems.

**Query Mediation System:** Mediator approach has been well studied for information integration from multiple sources, and many existing mediator systems include SIMS [5], Information Manifold [88], TSIMMIS [51], MedMaker [104], and a bioinformatics pipeline mediator platform proposed by Raschid et al [48]. In these approaches, the mediator provides users with seamless integrated views of the data
from heterogeneous sources and is capable of generating query plans for user queries. But query plans in mediator system are generated based on pre-specified rules or axioms, for example, the Mediator Specification (MS) rules used in TSIMMIS and MedMaker, the SIMS Axioms and the hand coded query pipeline for detecting alternative splicing of an organism’s genes [48]. However, our query planning algorithms traverse a graph space to find query plans based on query property and data source dependencies rather than pre-defined rules. Furthermore, existing mediator systems can only handle with simple selection and join like queries, but our system could deal with more complicated queries.

**Query Planning and Optimization with Limited Access Pattern:** Most work in query planning and Optimization with limited access pattern [108, 109, 123, 93, 31, 58, 133] is based on the System-R framework [116]. The existing work focuses on selection-project-join style conjunctive queries. Query plan is generated by searching a sub-plan search space to find the least cost plan. Our work is clearly distinct from the above work in the following aspects. First, we developed algorithms to handle with more advanced queries, such as aggregation and nested queries, which have specific requirement for a valid query plan. Second, in the deep web, base data is hidden behind query interfaces, so query planing and optimization cannot be performed based on relational table statistics as in System-R framework. We proposed our novel query merging and optimization techniques.

**Query Processing and optimization on Web Data Sources:** Query processing and optimization on web data sources arises much research efforts in literature [84, 17, 118, 119, 125, 20]. Kementsietsidis *et al* [84] proposed a system for optimizing exploratory queries over biological data sources. The main optimization they considered is to merge data sources among related exploratory queries. Srivastava *et al* [118] presented an algorithm which orders data sources in query plan so
as to minimize the query’s running time. In their system, they assume only one attribute can only be provided by one data source. Several query planning algorithms for web data sources are proposed in [17, 119, 125]. The query planning algorithms in [119, 125] are based on the *Steiner Tree algorithm*, which assumes that a valid query plan must be a *tree*. Braga *et al* [17] proposed a planning algorithm for multi-domain queries on the web, but this algorithm would cause the size of the query plan to grow exponentially. None of the above work considers query planning and optimization for complex queries including aggregation, grouping-by, and nested queries, which, however, is the focus of our work.

**Keyword Search on Relational Databases:** Recently, keyword search over relational databases has attracted a lot of attention [69, 68, 81, 94, 120]. In relational database keyword search, data tuples are represented as nodes, and foreign keys are represented as edges. In our case, since we don’t have the access to the database behind deep web data sources, our graph model is in the metadata level. Furthermore, except [120], all other work on relational database keyword search doesn’t consider aggregation queries. Although Tata *et al* [120] considered aggregation query in their SOAK system, no nested query is considered.

**Deep Web Crawling System:** Lately, there has also been a lot of work on mining useful information from the deep web [64, 65, 99]. However, the above work is focusing on database integration, schema matching and hidden data crawling. They don’t consider answering complex cross-source queries over multiple dependent data sources.
2.4 Work Related with Caching Mechanism for Query Answering Systems

We now compare our work about query caching mechanisms with existing work on a number of topics related to query optimization and query caching.

**Answering Queries Using Views:** Answering queries using views involves processing a query using previously defined materialized views, instead of accessing the original database. Much work has been done in this area [50, 54, 46, 38, 45]. Views related to the query are selected. The evaluation of the query is done by using the selected views and issuing the remainder query on the database relations. Query rewriting using views depends on result overlapping check, which cannot be directly applied to deep web context as the motivating example in Section 1 shows. Overall, these techniques have particularly focused on SQL-like queries over relational databases.

**Multi Query or Correlated Query Optimization:** Multiple queries that have common data is the main motivation for inter-query optimization. Sellis [117] proposes an algorithm which finds common tasks across queries and decomposes queries into smaller sub-queries. Sub-queries sharing common tasks and data can be executed together. Roy *et al.* [113] represent multiple queries in a single DAG, sharing subexpressions. A greedy algorithm picks a set of nodes from the DAG to be materialized and then finds the optimal plan for the given set of materialized nodes. Correlated queries are a special type of queries in which a sub-query can be executed iteratively. Rao and Ross [110] propose a strategy to reuse the invariants in such correlated queries to optimize the execution. In our work, we also try to identify the common tasks between the new query and previous cached queries, and use the
common tasks to find a detour for the new query plan. But in our scenario, users issue queries separately, and not in a batch.

**Caching Query Results from Previous Queries:** Deshpande et al. [44] propose caching small regions of the multidimensional space called *chunks*. Query results to be stored in the cache are broken up into chunks and the chunks are cached. When a new query is issued, chunks needed to answer that query are determined. Jensen et al. [73] present an architecture for query processing in the relational model extended with transaction time. Dependent on query containment condition, previous cached query results are reused. A client-side data-caching schema for relational databases with a central server and multiple clients, with concurrency control of the cached data, has also been proposed [82]. Query containment is checked and then the remainder query is sent to the database server to be processed separately. In our work, the focus is on reusing cached plans.

**Query Caching for Dynamic Web Content:** There are a number of efforts on optimizing query processing on data-intensive web sites on the server side [96, 22, 132, 43] or the client side [112, 82, 97, 25]. The caching mechanisms used on the server side cache the dynamically generated web pages, data, or XML fragments, with the goal of shrinking the query processing time on the server side [22, 96]. But, the network delivery time, a major component of the delay, is not reduced by the above techniques. Challenger et al. [25] present a new approach for caching dynamic web data focusing on synchronizing the data in the cache with the one in databases, so that the caches do not contain stale data. Robinson and Lowden [112] developed a system that supports the reuse of query results for syntactically unrelated queries using range constraints and the reuse is determined only by the cached data, and not by the previous queries or plans. Luo et al. [97] propose a proxy caching
mechanism for database-backend web sites on keyword search queries, which is *data-driven* caching. In their system, only one data source is considered. Our work is clearly distinct in considering multiple inter-dependent deep web data sources which can have data redundancy. Kou *et al.* [89] propose a cache model for deep web query which focuses on efficiently organizing the large amount of unstructured data from previously executed queries. This is different from our query-plan-driven caching.

### 2.5 Work Related with Supporting Fault Tolerance for Query Answering Systems

We now compare our work about supporting fault tolerance for query answering systems with existing work on a number of topics related to adaptive query processing and dynamic query optimization.

**Query Scrambling:** Traditionally, query processing approaches break down when unexpected delays happen. Query scrambling modifies query execution plans on-the-fly when delays are encountered during runtime [124, 3]. The major differences between their work and our approach is that first, query scrambling only reorders or creates operators, but does not use new data sources which are not in the original query plan. Second, the basic assumption in their query scrambling approach is that data source experiences temporary delays, as a result, by reordering the operators, the delays of one data source can be hidden by the execution of other sources.

**Adaptive Query Processing:** In data integration, often little is known in advance about sources properties, which necessitates the use of adaptive query processing techniques to adjust query processing on-the-fly. Ives *et al.* [71, 72] propose an adaptive query execution system Tukwila dealing with SQL like queries. The system exhibits adaptive behavior at two levels: between the optimizer and execution engine and
within the execution engine. Avnur and Hellerstein [6] propose a query processing mechanism called *eddies* which mainly deals with runtime query plan re-optimization for join operations. In our approach, we do not focus on query plan re-optimization based on updated data table statistics, instead, we are dealing with inaccessible data sources.

**Dynamic Query Optimization:** Traditional query optimizer assumes accurate knowledge of runtime parameters, but in reality, this assumption is often not justified. The mid-query re-optimization problem is addressed in [80]. Graefe and Ward [55] present a dynamic query optimization method by keeping multiple pre-computed alternative plans and choosing one plan dynamically based on query statistics. Similarly, Cole and Graefe [34] propose that during compile time, if two or more alternative plans are incomparable, all of them will be kept, and a better plan will be selected at the start-up time. Bodorik *et al.* [16] propose a strategy for correcting execution of distributed queries. The dynamic query optimization method proposed in [4] uses a query plan competition strategy in which multiple plans run at the same time to compete with each other and the winner will be continued with other plans terminated. Ioannidis *et al.* [70] proposed a parametric query optimization which attempts to generate one optimized query plan for each possible combinations of values for the resource parameters which are unknown at compile-time. Our work is different from above efforts in the following aspects. First, the problem addressed in our work is to adapt the query plan in order to deal with data source inaccessibility. Second, keyword search queries over deep web data sources is the focus of our work, not SQL like queries over relational databases or data warehouses.
2.6 Work Related with Mining Metadata of Deep Web Data Sources

We compare our work about mining metadata of deep web data sources with existing work on related topics, which are label extraction, retrieving data from hidden databases, and schema discovery for structured data.

Label Extraction and Template Mining: Deep web data source input interface label extraction [101, 135, 12] and output page template mining [130, 107, 134, 52] are two well studied areas. Several of the label extraction algorithms learn the layout of input interfaces using pre-specified rules [135]. Nguyen et al. [101] use a supervised learning strategy to train a two layer classifier to classify labels and non-labels. For learning the output page templates, most algorithms explore the repeated patterns appearing in the returned data records [130, 134] or identify the mapping between label and data value using heuristics [107]. The approach proposed by Gatterbauer [52] used model based method to learn web table layout. The above work focuses on extracting data or labels from a given web page, however, we are interested in wisely finding a list of output pages which could cover the complete output schema.

Retrieving Data from Hidden Databases: The work proposed in [23, 11, 21] focuses on retrieving data from hidden text data sources. This is different from our work because there is no metadata in text data sources, and the result from text data source for a keyword query is composed of a list of matched documents. Lidden et al [95] and Dasgupta et al [41] proposed systems on extracting data from hidden relational databases. Lidden’s system can extract all the data from a hidden data source and Dasgupta’s system is capable of finding a random data sample from a hidden data source. Madhavan et al [99] proposed a scalable system for sampling web pages generated from deep web data sources to build index for Google. However,
their systems are different from ours from the following aspects. First, they focus on extract the data, rather than metadata from hidden data sources. Second, they assume that given an input query, the returned output page has the complete metadata of the output schema. Third, Lidden’s approach mainly depends on a default query which has default value for each input schema attribute, while such default input query seldom appears for scientific data sources in our scenario.

**Schema Discovery for Structured Data:** Schema discovery is the problem of constructing a relational schema that best describes the extracted data. Most of the work in this area utilizes heuristics, ontology, or instance-based matching methods [18, 19]. Our work is clearly distinct from the above work, since we do not construct schemas.
CHAPTER 3
EFFECTIVE AND EFFICIENT SAMPLING METHODS
FOR DEEP WEB AGGREGATION QUERIES

A large part of the data on the World Wide Web (WWW) resides in hidden databases or the deep web. Executing structured, high-level queries on deep web data sources involves a number of challenges, several of which arise because query execution engines have a very limited access to data. In this chapter, we consider the problem of executing aggregation queries involving data enumeration on these data sources, which requires sampling. The existing work in this area (HDSampler) is based on simple random sampling. We observe that this approach has two critical limitations. First, it cannot obtain good estimates when the data is skewed, and second, because multiple deep web queries may be needed for selecting one sampling unit, it incurs high sampling costs. While there has been a lot of work on sampling skewed data, the existing methods are based on prior knowledge of data, and are therefore not applicable to hidden databases.

In this chapter, we present two prior-knowledge-free sampling algorithms, Adaptive Neighborhood Sampling (ANS) and sub-space based Two Phase adaptive Sampling (TPS), which allow an aggregation query to be answered with a high accuracy (even when there is a skew), and a low sampling cost. For this purpose, we have developed robust estimators for a number of common aggregation functions.
3.1 Motivation

A large portion of the data on the World Wide Web resides in the deep web. To access such hidden data, a user must issue queries through the input interfaces of deep web data sources. These queries are then executed over the hidden databases and answers are returned to the user as dynamically generated HTML pages. Deep web data sources are playing an important role in the modern society, impacting practically every Internet user. An early study on the deep web, conducted in year 2000, estimated that the public information in the deep web is 500 times larger than the surface web, with 7,500 Terabytes of data, across 200,000 deep web sites [15].

With the growth of the size and popularity of the deep web, it is becoming increasingly desirable to support structured, high-level queries over one or multiple deep web sources. Such support, however, involves many distinct challenges, depending upon the queries that one may support [63, 98]. In this chapter, we focus on aggregation queries that require data enumeration. These are distinct from the aggregation queries that can be answered directly using a data source’s input interface. For example, “find the hotel with the lowest price in Boston on 2/15/2010” can be directly answered using a travel web-site, and specifying city and date. In contrast, the following aggregation query requires data enumeration:

Example 1: Consider a deep web data source in the travel domain. Suppose a business student wants to study US aviation market, and for all major US airlines, he wants to obtain the average airfare from the US to Europe, across all flights in the next week.

To obtain the exact answer for this aggregation query, one needs to enumerate every pair of US and European city, and issue corresponding deep web queries. This can be extremely time-consuming, if not impossible. In general, finding the exact answer to an aggregation query that requires data enumeration is not practical due
to the following reasons. First, it is extremely hard to obtain all possible input values to enumerate the data. Second, deep web data is returned over a network, and thus, executing a large number of queries can be extremely time consuming. Some data sources even charge access costs. Third, many deep web sources limit the number of queries a particular IP address can issue, or the number of data records can be returned, to protect their data from being completely downloaded, or to disallow a denial-of-service attack.

Therefore, deep web aggregation queries can only be answered approximately, using sampling. We expect that users, such as the user in the example above, will be satisfied with a reasonably accurate, but approximate, answer. This, however, requires effective and efficient sampling. Even generating random samples from hidden databases can be challenging. Recently, Dasgupta et al have addressed this problem by developing HDSampler [41, 39], which is able to select a simple random sample (SRS) from hidden databases.

Our work is driven by the same motivation, but further considers the problem that a random sampling approach like HDSampler may not work effectively on many deep web data sources, because of the following reasons:

**Low Estimation Accuracy on Skewed Data:** HDSampler obtains a simple random sample from a hidden database. As is well known, SRS cannot provide good estimates on skewed data [103, 28]. For hidden databases, we do not know the data distribution in advance. The problems arising when the hidden data is skewed can be shown through the following example:

**Example 2:** We have a deep web source $D$ containing 10 records, $D = (1, 1, 1, 1, 1, 1, 1, 1, 10, 1000)$. We want to compute the average value for all data records in $D$. Clearly, the data distribution in $D$ is highly skewed. The true average value is 101.8. Suppose HDSampler takes a SRS of size 2. There are four possible samples we could
<table>
<thead>
<tr>
<th>HDSampler Sample</th>
<th>HDSampler Estimate</th>
<th>ACS Networks</th>
<th>ACS Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1=(1,1)$</td>
<td>1</td>
<td>${1},{1}$</td>
<td>1</td>
</tr>
<tr>
<td>$s_2=(1,10)$</td>
<td>5.5</td>
<td>${1},{10^*,1000}$</td>
<td>273.47</td>
</tr>
<tr>
<td>$s_3=(1,1000)$</td>
<td>500.5</td>
<td>${1},{10,1000^*}$</td>
<td>273.47</td>
</tr>
<tr>
<td>$s_4=(10,1000)$</td>
<td>505</td>
<td>${10^<em>},{1000^</em>}$</td>
<td>272.98</td>
</tr>
</tbody>
</table>

obtain: $s_1 = (1, 1)$, $s_2 = (1, 10)$, $s_3 = (1, 1000)$ and $s_4 = (10, 1000)$. The estimated averages obtained using these samples are shown in the second column of Table 3.1.

We observe that none of the samples yields a good estimate for the true average. This is because we are sampling data with equal selection probability. However, the skew causing data, which usually dominates the true average, only accounts for a small portion of the entire data. Therefore, these data records are often under-sampled.

**High Sampling Cost:** In the deep web scenario, sampling cost refers to the network transmission time for the queries issued to obtain the sample. As studied by Dasgupta et al [41, 39], to obtain one random sampled unit using HDSampler, multiple queries are often needed. Thus, obtaining a modest sized SRS using HDSampler would involve a high sampling cost.

As stated above, the limitations of SRS for skewed data are well known, and many other sampling methods have been developed to address these. For example, in the context of relational databases, many unequal probability sampling techniques have been developed, and have also been demonstrated to be effective with skewed data. Examples include density biased sampling [103], stratified sampling [30], sampling with outlier-indexing [28], histogram and wavelet based sampling [24, 106] and

| 31 |
random sampling with supplemental statistics [74]. However, all the above methods are built based on two assumptions. First, they require knowledge of the data distribution or key statistics, such as the size of the data strataums [30], outlier distribution [28] and sufficient data statistics [24, 106, 74]. Second, the sampler has access to the full dataset, i.e., it off-line scans the full data at least once, and obtains data statistics [103, 24, 106]. These two assumptions are reasonable in relational database or data warehouses. However, they do not hold true when there is a very limited access to hidden data, as in the deep web case. Furthermore, many deep web sources update data frequently. Even if we obtain partial data statistics ahead of time, they may not hold true at the time when we are responding to a user query.

Therefore, we need new techniques to support approximate aggregation queries over deep web data sources that may have skewed data. In this chapter, we have developed and evaluated two new adaptive sampling algorithms. They are the Adaptive Neighborhood Sampling (ANS) algorithm and the sub-space based Two Phase adaptive Sampling (TPS) algorithm. Both these algorithms have two advantages over HDSampler. First, both of them can provide accurate estimation for aggregation queries on skewed data, without requiring any knowledge of the hidden data distribution or other statistical information. Second, for data with a low skew, the TPS algorithm incurs significantly lower sampling costs compared to HDSampler. In this paper, we have considered four types of aggregation queries: Average (AVG), Summation (SUM), Maximal (MAX) and Minimal (MIN).

The basic ideas of the ANS and TPS algorithms are as follows.

**Handling Skewed Data in ANS and TPS:** Similar to other unequal probability sampling algorithms, ANS and TPS algorithms select a random sample biased towards the skew causing data. However, without any knowledge of the distribution or other statistics of hidden data, ANS and TPS algorithms sample skew causing data by
utilizing *neighborhood proximity*. We observe that skew causing data, which also has a *low frequency*, usually forms *clusters* with respect to certain attribute(s). For example, in household income data, very high income households likely have a *low frequency*, but they usually form clusters. For example, their household heads may have certain occupations such as doctor or lawyer, and/or the physical locations of these households may be clustered. In other words, skew causing income data has neighborhood proximity with respect to *occupation* and/or *location* attributes.

Unlike existing algorithms that handle each sampled unit independently, in ANS and TPS, we aggressively consider *neighboring units* for sampled units. For this, when *upper quantile* data (likely but not necessarily the skew causing data) is sampled, ANS and TPS algorithms explore its *neighborhood*, so as to increase the *selection probability* of the skew causing data.

**Lowering Sampling Cost in TPS:** In response to a deep web query, all data records matching with the query are returned. Because some data records match with more queries, the *selection probabilities* of the returned records are different. To enforce *uniform selection probability*, HDSampler uses *rejection sampling*. The result is that multiple queries may have to be used to select just one sampled unit. In TPS, to lower the sampling cost, we allow sampled units with unequal selection probabilities. Since the resulting selection probabilities are now non-uniform, we have developed novel robust estimators for the TPS algorithm based on the well-known Rao-Blackwell Theorem [111].

The overall contributions of our work can be summarized as follows.

1. We have developed two novel sampling algorithms and their theoretical analysis to approximately answer aggregation queries on deep web sources.

2. Without any prior knowledge of the data distribution, these algorithms yield
accurate estimation on skewed data. In addition, the TPS algorithm has lower sampling costs.

3. Our experiments show that for data with a moderate or a large skew, ANS and TPS yield more accurate estimates, outperforming HDSampler by a factor of 4 on the average. For data with a small skew, TPS method only incurs one-third of the sampling cost of HDSampler.

### 3.2 Background

In this section, we first review HDSampler, a recently developed system for obtaining simple random samples from structured hidden databases [41, 39]. We also state the estimators for simple random samples obtained by HDSampler. Then, we give an overview of *adaptive cluster sampling*, an existing method for sampling skewed datasets [122].

#### 3.2.1 HDSampler and Its Estimators

HDSampler performs a *random walk* through a tree. Each level of the tree represents an input attribute and the database tuples exist at some of the leaves. Each path from root to a leaf represents a specific assignment of values to input attributes, and only some of the leaves correspond to actual tuples in the database. HDSampler performs a random *drill-down* by adding randomly selected input attributes as query predicates, until a tuple is selected. Because a tuple at higher level of the tree is more likely to be sampled than one at lower levels, the retrieved tuples are post-processed by *rejection sampling* to enforce equal selection probability.

Suppose we have a data source $D$ comprising $N$ data records, and the attribute of interest is $Y$. The values of $Y$ in $D$ can be represented as $\{y_1, y_2, \ldots, y_N\}$. Suppose $S = \{y_1, y_2, \ldots, y_n\}$ is the SRS selected using HDSampler from $D$. Here, $n$ is the
sample size. Each data record in $S$ is a *sampled unit*. Using $S$, the *estimates* of average, sum, and the maximum and minimum values of $Y$ can be computed as:

$$
\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}, \hat{t} = \bar{y}N, \hat{y}_{\text{max}} = \text{Max}_{i=1\ldots n}\{y_i\} \text{ and } \hat{y}_{\text{min}} = \text{Min}_{i=1\ldots n}\{y_i\}
$$

They are the *estimators* of HDSampler.

To evaluate the accuracy of an estimator, we use the metric *Absolute Error Rate* (AER). For a variable with true value $\theta$ and estimated value $\hat{\theta}$, the AER of the estimator $\hat{\theta}$ is

$$
\text{AER}(\hat{\theta}) = \frac{|\hat{\theta} - \theta|}{\theta}.
$$

### 3.2.2 Adaptive Cluster Sampling

The basic idea of the ANS and TPS algorithms that we are describing in this paper is inspired by *Adaptive Cluster Sampling (ACS)* proposed by Thompson in the early 90’s [122]. ACS was originally motivated by the problem of sampling *rare* and *clustered* population, such as rare animals, or a species like shrimp, which is in high abundance in certain spatial areas. We may want to estimate the total number of such animals within a certain area, which is further partitioned into *grids*. The animal population is highly skewed, implying that most grids do not have any animals, whereas, some areas have a high density. In addition, we are likely to see two other key features. First, it is hard to know their distribution in advance over the entire area. Second, they often form *clusters*. Thus, if we find them in one spatial area, we are more likely to find them in neighboring areas. With such properties, simple random sampling (SRS) usually does not perform well.

ACS is proposed for such cases, and works in two steps as follows. In the first step, a SRS is selected, denoted as the *initial sample*. If any unit (grid) in the initial sample satisfies a pre-defined *limit condition*, such as the number of animals in the unit is greater than a certain threshold, the unit is called a *unit of interest*. Then, in the second step, for each such unit of interest in the initial sample, its *neighboring*
units are further sampled. If any neighboring unit also satisfies the limit condition, it is added to the sample, and this process is repeated. The idea behind this approach is that because animals often form clusters, if we examine the neighboring units of a unit of interest, we are likely to obtain more units of interest into the sample. The neighborhood in this approach is usually determined by spatial adjacency. A unit of interest with all its neighboring units that also satisfy the limit condition form a network. The size of the network is the number of units in the network. For a unit in the initial sample which does not satisfy the limit condition, a network of size 1 is formed.

In ACS with an initial sample of size \( n \), we will eventually have \( n \) networks, which can be represented as \( S = \{s_1^*, s_2^*, \ldots, s_n^*\} \), where \( s_i^* \) is the \( i^{th} \) network. If we use \( \bar{y}_i^* \) to represent the average value of the attribute of interest in the network \( i \), an unbiased estimator for the population average is \( \bar{y} = \frac{\sum_{i=1}^{n} \bar{y}_i^*}{n} \) [122].

Let us revisit the Example 2 from Section 3.1, and consider the application of ACS. Here, we define the data records with values no less than 10 as the units of interest, and neighborhood is determined based on spatial adjacency, implying that data records with values of 10 and 1000 are neighbors. The ACS network and ACS estimated averages are shown in the last two columns of Table 3.1. In the “ACS networks” column, the unit(s) enclosed within brackets \( \{} \) represent(s) a network. In each network, the unit with a * is the unit of interest (whose value no less than 10) in the network, and other units in the network are neighboring units. We observe that for the initial samples \( s_2, s_3, \) and \( s_4 \), ACS gives much better estimates than HDSampler, which is based on simple random sampling. This is because in these three samples, the skew causing data is included. However, if the initial sample is \( s_1 \), the ACS estimate is the same as HDSampler.

From this example, ACS appears like a promising approach for sampling hidden
databases. However, ACS also has three limitations that will have to be addressed to make it applicable to the deep web. First, as shown in the Example 2, the performance of ACS depends on the initial sample. If there is no unit of interest in the initial sample, ACS degrades to simple random sampling. Second, ACS has not been developed or analyzed for cases with a sampling cost limit. In comparison, in the deep web scenario, since many queries are executed on-the-fly, i.e. in an interactive setting with a user, it is necessary to consider a cost limit. This could further include: 1) how many units in the initial sample are allowed? and 2) how many neighboring sampled units are allowed? The third challenge in applying ACS is that the initial sample in ACS is selected as a simple random sample. As stated in Section 3.1, obtaining SRS on a deep web source has a high cost.

We propose an adaptive neighborhood sampling algorithm which overcomes the first two limitations of ACS. We also develop a two-phase adaptive sampling algorithm which addresses the first limitation of ACS and also provides a practical solution for the third limitation.

3.3 New Sampling Algorithms

In this section, we first state several definitions and assumptions to formulate the sampling problem we will be addressing. Then, we describe the adaptive neighborhood sampling (ANS) and sub-space based two phase adaptive sampling (TPS) algorithms in details. Along with the description of the two algorithms, we also develop estimators for each of them, for computing the average (AVG) and summation (SUM) aggregation functions.

Towards the end of this section, we develop estimators for the maximum (MAX) and minimum (MIN) functions. We have also extended our current estimators to the
cases where we have multiple correlated data sources, or where queries have multivariate variables.

3.3.1 Problem Formulation

Consider a deep web data source $D$ with $N$ tuples $\{r_1, r_2, \ldots, r_N\}$ over a set of $m$ attributes $\text{Attr} = \{A_1, A_2, \ldots, A_m\}$. We assume that $N$ can be obtained from data source documentation, if we need to estimate the SUM. Attributes can be numerical or categorical. A subset of the attribute set $\text{Attr}$ is the input attribute set of $D$, and is denoted as $\text{IN} = \{A_1, A_2, \ldots, A_s\}$. The attribute of interest is a numerical attribute $A_q$, where $A_q \in \text{Attr}$ and $A_q \notin \text{IN}$. For an aggregation query on $A_q$ which requires data enumeration, we select a sample $S \subseteq D$, to estimate the result $\text{AGG}(A_q)$.

Unit of Interest

For a unit $r_i$ sampled from $D$, we use $y_i$, which is the value of $r_i$’s attribute of interest, to represent $r_i$. Then, we denote a unit $y_i$ in a sample $S = \{y_1, y_2, \ldots, y_n\}$ to be a unit of interest, if $y_i$ satisfies the limit condition $y_i > \eta$. Here, $\eta$ is the upper limit value that can be determined based on domain knowledge. For dataset with negative values on the attribute of interest, the limit condition $y_i \leq \eta'$ is also considered.

Data Record Neighborhood

In ACS, the neighborhood is determined based on spacial adjacency. Here, we define neighborhood in the deep web scenario. For a data source $D$ with $m$ attributes, we consider $D$ as an $m$-dimensional space. Each data record $r$ is a point in this $m$-dimensional space, denoted as $r = \{a_1, \ldots, a_s, y, \ldots, a_m\}$. $a_i$ is the value of the attribute $A_i$ in data record $r$. The first $s$ values correspond to the input attributes, and $y$ is the value of the attribute of interest. Intuitively, the neighborhood of $r$ is the set of data records that are adjacent to $r$ with respect to the input attributes.
Specifically, we take data records that have a different value for at most one of the input (or the first \(s\)) dimensions. This is denoted as

\[
\text{Neighborhood}(r) = \{r' = \{a_1, \ldots, \text{neighborhood}(a_j), \ldots, a_s, \ast, \ldots, \ast\} | 1 \leq j \leq s\}
\]

where, \(\ast\) denotes values for non-input attributes of \(r'\). If \(A_j\) is a numerical attribute, \(\text{neighborhood}(a_j) = \{a_j, a_j \pm \text{unit length}\}\), where \text{unit length} is chosen depending upon the scale of \(A_j\).

For a categorical attribute, we assume its values can have categorical neighborhoods, through knowledge of the domain. For example, suppose we have the following values for the occupation attribute: doctor, lawyer, cleaner, and cashiers. In such a case, we can form two categorical neighborhoods, one for high wages, which includes doctor and lawyer, and the other for low wages. Thus, in computing the neighborhood, if \(A_j\) is a categorical attribute, \(\text{neighborhood}(a_j)\) contains all values in the categorical neighborhood of \(A_j\) as described above.

Thus, a neighboring unit \(r'\) of \(r\) falls in either of the following two types: 1). Immediate Neighbor, \(r'\) has the same input attribute values as \(r\) but different values on other attributes, 2). Collateral Neighbor, \(r'\) has at most one input attribute value different from that of \(r\), but this value is still adjacent to the corresponding value for \(r\). As an example, if we have a data record in a 2-dimension space \(r = (a_1 = 2, y = 4)\), then the neighborhood of \(r\) contains \((2, \ast), (1, \ast), (3, \ast)\), of which the first one is an immediate neighbor and the other two are collateral neighbors.

Neighborhood Proximity

The ACS algorithm utilizes the property that the sampling units are always clustered to achieve good performance. In most real data sets, neighborhood proximity is a common phenomenon as we mentioned in Section 3.1. Consider data records \(r, r'\)
and $r^*$, where $r'$ is one of $r$’s neighbors, and $r^*$ is an arbitrary record not neighboring with $r$. Neighborhood proximity is a property that makes it likely that

$$|y_r - y_{r'}| < |y_r - y_{r^*}|$$

where, $y$ is the value of the attribute of interest. Neighborhood proximity is different from attribute correlation.

**Direct Samples and Associated Samples**

In response to an input query, an answer set $W = \{r_1, \ldots, r_l\}$ is returned. Using a particular sampling algorithm, suppose one unit $r_g \in W$ is selected. We consider $r_g$ to be a *direct sampled unit*. All other returned tuples in $W - \{r_g\}$ are considered as *associated sampled units*. The sampling cost of $r_g$ is the network cost of the entire answer set $W$. Therefore, we could consider the associated sampled units are being available free of any additional sampling costs.

In Section 3.3.1, two types of neighboring records were described. Among these, *immediate neighbors* of $r_g$ are the ones that share the same input values as $r_g$, i.e. they are part of the same answer set as $r_g$. As a result, immediate neighbors are associated sampled units. *Collateral neighbors* of $r_g$ have different input values from $r_g$, so collateral neighbors, if obtained, will be direct sampled units. In our algorithms, we will be exploiting the available neighbors of the units of interest. Since immediate neighbors are free of sampling cost, this provides our algorithms a critical advantage, with the results that their sampling costs are lower than that of HDSampler.

### 3.3.2 Adaptive Neighborhood Sampling (ANS)

Recall that in Section 3.2.2, we had pointed out three limitations of the adaptive cluster sampling (ACS). In this section, we describe the adaptive neighborhood sampling (ANS) algorithm to address the first two limitations of ACS, which are first,
the performance of ACS depends on the initial sample, and second, ACS has not been developed or analyzed for cases with a time-limit, or a *termination* condition on sampling.

The ANS algorithm selects sampled units in two steps. In the first step, ANS selects an SRS as the *initial sample* using HDSampler. In the second step, for each sampled *unit of interest* $r_i$ in the initial sample, the units *neighboring* $r_i$ are further sampled to form the network corresponding to $r_i$. Formally, the method is shown as Algorithm 3.3.1. We detail these two steps of the ANS algorithm in the following paragraphs.
Algorithm 3.3.1: ANS:\((k, n, \eta)\)

/*The first step, obtain initial sample*/

\(\text{of\_interest} = 0\) /*number of units of interest in initial sample*/

\(\text{initialsize} = 0;\) /*size of initial sample*/

/*consider the two termination rules*/

\textbf{while} initialsize < \(n\) AND of\_interest < \(k\)

\(r = \text{HDSampler()}\) /*find a random sampled unit*/

add \(r\) to the initial sample, \(\text{initialsize} + +\)

\textbf{if} \(r\) is a unit of interest

\(\text{of\_interest} + +;\)

/*the second step, explore the neighborhood*/

\textbf{foreach} unit of interest \(r\) in the initial sample

\(\text{networksize} = 0\)

\(\text{Neighbors} = \text{FindNeighbor}(r)\) /*find the neighboring units of \(r\/*/\)

\textbf{foreach} \(nr \in \text{Neighbors}\)

\textbf{if} \(nr.y > \eta\)

/*\(nr\) satisfies the limit condition*/

add \(nr\) to the network of \(r\) \(\text{networksize} + +\)

add \(\text{FindNeighbor}(nr)\) to \(\text{Neighbors}\)

/*adding neighbors in a recursive manner*/

/*adaptively adjust the value of \(\eta\/*/\)

\textbf{if} \(\text{networksize} > \text{upperthreshold}/*too many neighboring units added*/\)

\(\eta = \eta + \text{unit}\_\text{length}\)

\textbf{else}

\textbf{if} \(\text{networksize} < \text{lowerthreshold}/*too few neighboring units added*/\)

\(\eta = \eta - \text{unit}\_\text{length}\)
Selecting Initial Sample in ANS

The first limitation of ACS is that if there is no unit of interest in the initial sample, ACS degrades to random sampling. To overcome this limitation, an intuitive solution is to ensure that a certain number of units of interest, say $k$, are included in the initial sample in the ANS algorithm. For this, we perform simple random sampling using HDSampler until $k$ units of interest are observed in the initial sample. However, in order to achieve $k$ units of interest in the initial sample, the final size of the initial sample can be arbitrarily high. Therefore, in the deep web scenario, we must include appropriate termination rules for obtaining the initial sample.

Consider a query answering system interacting with a user. The user may have a time or cost-limit in mind, implying that an approximate answer to his query should be obtained within the time limit. Based on profile or history information, we may know the average network delay for one sampled unit. Similarly, we may also know the average relative ratio between the size of the initial sample and the size of the neighborhood sample. Then, given a user cost limit $C$ (in terms of time), we could decide the corresponding maximal size of the initial sample allowed, which is denoted as $n$.

ANS Termination Rules for Obtaining Initial Sample: In ANS, to obtain the initial sample, we use HDSampler to select a sequence of random sampled units until either of the following two termination rules are satisfied: 1) we have observed $k(k_i 1)$ number of units of interest in the current initial sample; or, 2) the size of the current initial sample reaches $n$. The first termination rule implies that if we have already obtained the desired number of units of interest in the initial sample, we could stop sampling even before reaching the limit $n$. The second rule implies that if we cannot select $k$ units of interest in the initial sample within the limit $n$, we need to just proceed with the initial sample already obtained.
Selecting Neighborhood Sample in ANS

Selecting the neighboring units (immediate or collateral neighbors) of a unit of interest is an important aspect of the ANS algorithm. As described in Section 3.3.1, the unit of interest is determined based on the upper limit value $\eta$. For a sampled unit $r$, if the value of the attribute of interest of $r$ is greater than $\eta$, $r$ is a unit of interest, and $r$’s neighbors that also satisfy the limit condition are also included in the final sample. The original adaptive clustering sampling algorithm assumes a fixed $\eta$ value for the entire sampling process.

Clearly, the choice of the value of $\eta$ is important for the process. For example, the value of $\eta$ impacts the number of neighboring units that can be added to the final sample. If the value of $\eta$ is too small, too many neighboring units will be added to the final sample. This could involve extra sampling costs since collateral neighbors are direct sampled units and obtaining them requires additional queries. If the value of $\eta$ is too large, we end up with too few neighboring units, which in turn could negatively impact the estimator. Therefore, during sampling with the ANS algorithm, we should adaptively adjust $\eta$ to control the number of neighboring units added to the final sample.

Thus, after obtaining the initial sample as described in Section 3.3.2, for each unit of interest in the initial sample, we find its corresponding network by adding its neighboring units that satisfy the limit condition into the sample. We repeat the process till no more neighbors can be added. After neighborhood exploration for a unit of interest $r_i$ in the initial sample, if the current $\eta$ has introduced too many neighboring units into the network of $r_i$, i.e., the size of the network exceeds a pre-specified threshold, we will increase the value of $\eta$. Thus, for other units of interest left in the initial sample, the number of neighborhood units added to their networks
could be decreased. If the current $\eta$ introduces very few neighboring units into the network of $r_i$, we will decrease the value of $\eta$.

### 3.3.3 Estimators for ANS Algorithm

In this section, we develop the estimators of the AVG and SUM estimates for the ANS algorithm. Throughout this section, we assume that the total number of records in the data source is $N$, the number of sampled units in the initial sample is $n$, and the number of units of interest in the initial sample is $l$. Following the idea of adaptive cluster sampling introduced in Section 3.2.2, the ANS algorithm obtains a set of sampled networks, which is $S = \{s_1^*, s_2^*, \ldots, s_n^*\}$, where $s_i^*$ is the $i^{th}$ network formed by the $i^{th}$ unit in the initial sample.

Developing such estimators involves several challenges, since the algorithm can stop with either of the two termination conditions, and the value of $\eta$ can be dynamically modified. Thus, we first develop the estimators for the case that a fixed $\eta$ value is used in the ANS algorithm. Then, we modify the estimators for the case when different $\eta$ values are used in the algorithm.

#### Estimators for Fixed $\eta$

In designing the estimators for the ANS algorithm with a fixed $\eta$ value, we need to consider two cases, corresponding to the two termination rules. We first design estimators for the sample selected based on each termination rule independently, then we combine them together and provide theoretical analysis.

**Estimator for AVG Under 1st Termination Rule:** Using the first termination rule, the initial sample $S = \{s_1^*, s_2^*, \ldots, s_n^*\}$ contains $k$ units of interest and the size of the initial sample is smaller than or equal to the limit $n$. An intuitive AVG estimator will be the *sample average estimator*, which was also introduced in Section 3.2.2.
Specifically, we will compute $\bar{y} = \frac{\sum_{i=1}^{n} y_i^*}{n}$, where $y_i^*$ is the average value of the attribute of interest in the network $s_i^*$. However, under the first termination rule, the size of the initial sample depends on when the $k^{th}$ unit of interest is sampled. Thus the initial sample size $n$ is not fixed. As a result, the above estimator, designed for the cases with fixed sample sizes, is biased [90]. Therefore, we have developed an alternative estimator.

In the data source $D$, suppose the total number of units of interest in the entire data set is $M$. Then, an unbiased estimator for the population average $\bar{y}$ for the attribute of interest using the sample obtained under the first termination rule is

$$\tilde{y}_{tr1} = \frac{\hat{M} \bar{y}_M^*}{N} + \frac{(N - \hat{M})}{N} \bar{y}_{N-M}^*$$

where, $\bar{y}_M^* = \frac{\sum_{i \in S_M} y_i^*}{k}$, $S_M$ is the set of networks formed by the $k$ units of interest in the initial sample. $\bar{y}_{N-M}^* = \frac{\sum_{i \in S_{N-M}} y_i^*}{k}$, $S_{N-M}$ is the set of networks formed by ordinary units (i.e. units not of interest) in the initial sample. $\hat{M}$ is the estimator of $M$, which can be approximated by $\hat{M} = \frac{N(k-1)}{n-1}$ [92].

Lemma 3.3.1. The estimator $\tilde{y}_{tr1}$ is an unbiased estimator under the first termination rule.

Proof. In the data source $D$, suppose the total number of units of interest in the entire data set is $M$, and the total number of data records in $D$ is $N$. Then, we know that $\beta = \frac{M}{N}$. We use $U_{N-M}$ to represent all ordinary units (i.e., units not of interest) in the entire data set, and use $U_M$ to represent all units of interest in the entire data set. Let $f_i$ to be the number of times the $i^{th}$ unit from the set $U_{N-M}$ appears in the sample, and we have $f_i \sim \text{binomial}(n - l, \frac{1}{N-M})$. $g_i$ is the number of times the $i^{th}$
unit from the set $U_M$ appears in the sample, and we have $g_i \sim \text{binomial}(l, \frac{1}{M})$. Then the expected value of $\hat{y}_{tr1}$ can be obtained as

$$E[\hat{y}_{tr1}] = E[\hat{\beta}\hat{y}_1^* + (1 - \hat{\beta})\hat{y}_2^*] = E\left[\frac{\hat{M}\hat{y}_1^*}{N} + \frac{(N - \hat{M})\hat{y}_2^*}{N}\right]
= E\left[\frac{N - \hat{M}}{N - M} \frac{\sum_{i \in U_{N-M}} y_i}{N} + \frac{\hat{M}}{M} \frac{\sum_{i \in U_M} y_i}{N}\right] = \bar{y}$$

$\hat{y}_{tr1}$ is an unbiased estimator under the first termination rule.

**Estimator for SUM Under 1st Termination Rule:** Similarly, an unbiased estimator for the population total $\hat{t}_{tr1}$ for the attribute of interest using the sample obtained under the first termination rule is

$$\hat{t}_{tr1} = \hat{M}\hat{y}_M^* + (N - \hat{M})\hat{y}_{N-M}^*$$

**Estimator for AVG Under 2nd Termination Rule:** Using the second termination rule, the initial sample $S = \{s_1^*, s_2^*, \ldots, s_n^*\}$ has a fixed sample size $n$. As a result, the sample average estimator, which was also introduced in Section 3.2.2 can be used here as an unbiased estimator. An unbiased estimator for the population average $\bar{y}$ for the attribute of interest using the sample obtained under the second termination rule is

$$\bar{y}_{tr2} = \frac{1}{n} \sum_{i=1}^{n} \bar{y}_i^*$$

where $\bar{y}_i^*$ is the average value of the attribute of interest in the network $s_i^*$.

**Estimator for SUM Under 2nd Termination Rule:** Similarly, an unbiased estimator for the population total $\hat{t}_{tr2}$ for the attribute of interest using the sample obtained under the second termination rule is

$$\hat{t}_{tr2} = \frac{N}{n} \sum_{i=1}^{n} \bar{y}_i^*$$
Combined Estimator for AVG: We use \( l \) to represent the total number of units of interest in the initial sample. The estimator for the population average \( \bar{y} \) for ANS algorithm considering two termination rules with fixed \( \eta \) value is as follow.

\[
\bar{y}_{\text{ANS}} = \begin{cases} 
\frac{M\bar{y}_M}{N} + \frac{(N-M)\bar{y}_{N-M}}{N} & l = k, \\
\frac{1}{n} \sum_{i=1}^{n} \bar{y}_i & l < k. 
\end{cases}
\]

When considering the estimators under each termination rule independently, both of them are unbiased estimators. However, if the two termination rules are jointly considered, the combined estimator is biased. The intuitive explanation of the bias is as follows. When the two termination rules are considered together, the size of the initial sample from the ANS algorithm is not fixed, whereas, one part of the combined estimator requires fixed sample size. Lemma 3.3.2 computes the expectation of \( \bar{y}_{\text{ANS}} \) showing the bias of this estimator.

**Lemma 3.3.2.** The expectation of \( \bar{y}_{\text{ANS}} \) is \( E[\bar{y}_{\text{ANS}}] = \bar{y} \times Pr[l \leq k] \), where \( Pr[l \leq k] \) is the probability that the actual number of units of interest in the final initial sample being smaller or equal to \( k \).

**Proof.** The expectation of \( \bar{y}_{\text{ANS}} \) can be computed as follows.

\[
E[\bar{y}_{\text{ANS}}] = E[(\frac{M\bar{y}_M}{N} + \frac{(N-M)\bar{y}_{N-M}}{N})I[l = k] + \bar{y}_n I[l < k]]
\]

We first consider the first part of the above expectation.

\[
E[(\frac{M\bar{y}_M}{N} + \frac{(N-M)\bar{y}_{N-M}}{N})I[l = k]]
\]

\[
= E[I[l = k]E[(\frac{M\bar{y}_M}{N} + \frac{(N-M)\bar{y}_{N-M}}{N})|l]]
\]
where

\[
E\left[ \frac{\hat{M} \bar{y}_M^*}{N} + \frac{(N - \hat{M}) \bar{y}_{N-M}^*}{N} \right] = \frac{N - M}{N - M} \sum_{i \in U_{N-M}} y_i + \frac{\hat{M}}{N} \sum_{i \in U_M} y_i^*
\]

so we get

\[
E\left[ \frac{\hat{M} \bar{y}_M^*}{N} + \frac{(N - \hat{M}) \bar{y}_{N-M}^*}{N} \right] = E\left[ \left\{ \frac{N - \hat{M}}{N - M} \sum_{i \in U_{N-M}} y_i + \frac{\hat{M}}{N} \sum_{i \in U_M} y_i^* \right\} I[l = k] \right]
\]

\[
= \sum_{i \in U_{N-M}} y_i \frac{1}{N - M} Pr[l = k]
\]

\[
+ E\left[ -\frac{1}{N - M} \sum_{i \in U_{N-M}} y_i + \frac{1}{M} \sum_{i \in U_M} y_i^* \right] \times \frac{\hat{M}}{N} I[l = k]
\]

where

\[
E\left[ \frac{\hat{M}}{N} I[l = k] \right] = \frac{M}{N} Pr[l = k]
\]

So, we have

\[
E\left[ \frac{\hat{M} \bar{y}_M^*}{N} + \frac{(N - \hat{M}) \bar{y}_{N-M}^*}{N} \right] I[l = k]
\]

\[
= \frac{1}{N} \left\{ \sum_{i \in U_{N-M}} y_i \frac{N}{N - M} Pr[l = k] \right\}
\]

\[
+ (\sum_{i \in U_M} y_i^* - \frac{M}{N - M} \sum_{i \in U_{N-M}} y_i) \times Pr[l = k]
\]

(1)

For the second part of the above expectation, we have

\[
E[y_i^* | l < k]
\]

\[
= E\left[ \frac{1}{n} \left( \sum_{i \in U_{N-M}} y_i f_i + \sum_{i \in U_M} y_i^* g_i \right) | l < k \right]
\]

where \( f_i \) is the number of times the \( i^{th} \) unit from the set \( U_{N-M} \) appears in the sample, and \( f_i \sim \text{binomial}(n - l, \frac{1}{N-M}) \). \( g_i \) is the number of times the \( i^{th} \) unit from the set \( U_M \) appears in the sample, and \( g_i \sim \text{binomial}(l, \frac{1}{M}) \).
As a result, we have

\[
E[I[l < k]E[y^*_n | l]]
\]

\[
= E\left[ \frac{1}{n} \left( \sum_{i \in U_{N-M}} y_i \frac{n - l}{N - M} + \sum_{i \in U_M} y^*_i \frac{l}{M} \right) I[l < k] \right]
\]

\[
= E\left[ \sum_{i \in U_{N-M}} y_i \frac{1}{N - M} I[l < k] + \frac{1}{n} \left( \frac{1}{M} \sum_{i \in U_M} y^*_i - \frac{1}{N - M} \sum_{i \in U_{N-M}} y_i \right) I[l < k] \right]
\]

And we have

\[
E[I[l < k]] = (n)pPr[l < k]
\]

So we have

\[
E[y^*_n | l < k] = \sum_{i \in U_{N-M}} y_i \frac{1}{N - M} Pr[l < k]
\]

\[
+ \frac{1}{N} \left\{ \left( \sum_{i \in U_M} y^*_i \frac{M}{N - M} \right) Pr[l < k] \right\}
\]

(2)
Add equation 1 with equation 2, we have the final expectation of the estimator as

\[
E[\bar{y}_{ANS}] = \frac{1}{N} \left\{ \sum_{i \in U_{N-M}} y_i \frac{N}{N-M} Pr[l = k] + \sum_{i \in U_{N-M}} y_i \frac{N}{N-M} Pr[l < k] + \sum_{i \in U_M} y_i^* Pr[l = k] + \sum_{i \in U_M} y_i^* Pr[l < k] - \sum_{i \in U_{N-M}} y_i \frac{M}{N-M} Pr[l = k] - \sum_{i \in U_{N-M}} y_i \frac{M}{N-M} Pr[l < k] \right\} + \bar{y} Pr[l \leq k]
\]

As a result, the estimator \( \bar{y}_{ANS} \) is biased, and the bias factor is \( Pr[l \leq k] \).

Lemma 3.3.3 shows that if \( k \) is small, the bias of \( \bar{y}_{ANS} \) is very small. In our experiment, we show that for relatively small \( k \) (such as \( k = 8 \)), the biased estimator has good performance.

**Lemma 3.3.3.** For a relatively large data set, if \( k \) is small, the bias factor in \( \bar{y}_{ANS} \) tends to be 1, \( Pr[l \leq k] \to 1 \). Therefore, the bias of the estimator \( \bar{y}_{ANS} \) is very small.

**Proof.** We give a quick sketch of the proof here. The variable \( l \), the actual number of units of interest in the final initial sample, follows the hypergeometric distribution \( l \sim \text{hypergeometric}(N, M, n) \). From the property of hypergeometric distribution, we know that, When \( M \ll N \) and \( k \ll M \), we have the probability \( Pr[l \leq k] \to 1 \). This can be shown as follows.
Pr\left[l \leq k\right] = \sum_{k=0}^{l} \binom{k}{M} \binom{n-k}{N} \frac{M \ll N, k \ll M}{\frac{n}{N}} = 1

In our scenario, since the skew causing data usually only accounts for a very small portion of the entire data, we have $M \ll N$. Furthermore, because the skew causing data is hard to be sampled, we have $k \ll M$ for a modest sized sample. Both of the above two conditions hold in our scenario, and thus, the result from Lemma 3.3.3 can be applied to our case.

In statistics, a biased estimator is not necessarily a poor or unacceptable estimator. If the bias is not severe, at least under some realistic conditions, and the biased estimator has other desirable properties, we still favor a biased estimator over an unbiased estimator without the same desirable properties. Lemma 3.3.3 shows that under certain conditions, which often hold in practice, the bias of the combined estimator described above is very small. This is further validated by our experiments. Furthermore, the combined estimator has two very desirable properties. First, it supports two practical termination conditions, whereas the original estimator for adaptive clustering sampling introduced in Section 3.2.2 does not consider any termination condition. Second, as we will show through our experiments, the estimation accuracy of the combined ANS estimator is better than that of the SRS estimator used by HDSampler.

**Combined Estimator for SUM:** Similarly, The estimator for the population total $\hat{t}_{\text{ANS}}$ for ANS algorithm considering two termination rules with fixed $\eta$ value is

\[
\hat{t}_{\text{ANS}} = \begin{cases} 
\hat{M} \hat{y}_M^* + (N - \hat{M})\hat{y}_{N-M}^* & l = k, \\
\frac{N}{\pi} \sum_{i=1}^{n} \hat{y}_i^* & l < k.
\end{cases}
\]

$\hat{t}_{\text{ANS}}$ has the same property as $\bar{y}_{\text{ANS}}$.  

\section{Conclusion}

This work presents an efficient and accurate method for estimating the total of skewed data using the ANS algorithm. The key contributions include:

1. Development of a novel estimator that combines ANS sampling with a skew correction technique.
2. Derivation of theoretical bounds on the bias and variance of the proposed estimator.
3. Experimental validation showing significant improvements over traditional sampling methods, especially in skewed data scenarios.

The proposed method is particularly useful in applications where data skew is common, such as in social media analytics, healthcare data analysis, and financial market studies. Further work could explore the integration of this method with machine learning algorithms to enhance predictive accuracy.
Estimators for Multiple $\eta$

When we use different $\eta$ values in the ANS algorithm, the final sample is post-stratified into a list of sub-sample spaces based on different $\eta$ values, i.e., each stratum contains the units sampled using one $\eta$ value. Therefore, we should modify the estimators developed for fixed $\eta$ using post-stratification. Suppose we have $H$ post-stratified strata for a sample of size $n$ from the ANS algorithm. The estimators are modified as follows:

**Estimator for AVG with Multiple $\eta$:** The estimator of the population average using post-stratification in the ANS algorithm is

$$\bar{y}_{\text{post}} = \frac{1}{n} \sum_{h=1}^{H} \frac{n_h}{n} \bar{w}_h$$

where $n_h$ is the total number of units in the initial sample that belongs to the $h^{th}$ stratum. $n_h$ can be obtained after post-stratification. $\bar{w}_h$ is the estimated average value using all the sampled units in the $h^{th}$ stratum. Since all the sampled units in one stratum share the same $\eta$ value, $\bar{w}_h$ can be computed using the estimator $\bar{y}_{\text{ANS}}$ shown earlier. The fraction $\frac{n_h}{n}$ shows the percentage of the sampled units in the initial sample falls into the $h^{th}$ stratum, and it is used as the weight for the estimated average from the $h^{th}$ stratum.

**Estimator for SUM with Multiple $\eta$:** Similarly, the estimator of the summation using post-stratification in the ANS algorithm is

$$\hat{t}_{\text{post}} = \frac{1}{n} \sum_{h=1}^{H} n_h \bar{w}_h$$

3.3.4 Sub-Space Based Two Phase Adaptive Sampling (TPS)

In the ANS algorithm, we still need to use HDSampler to select an initial sample, which can result in high sampling costs. In this section, we present another algorithm, which is the sub-space based Two Phase adaptive Sampling (TPS) algorithm.
The basic idea of this algorithm is as follows. We consider all data records in a data source \( D \) as being partitioned into a set of sub-spaces based on the value of input attributes. Each sub-space comprises the data records returned by inputs where every categorical attribute has the same value, and every numerical attribute has the value within a certain range. For example, if a data source \( D \) has a single categorical input attribute with 5 distinct values, we could consider the data records in \( D \) are being partitioned into 5 sub-spaces.

The key observation is that all or a large fraction of the records from a sub-space can be retrieved with a single query over a deep web source. Most data sources even allow a query where a range can be specified for a numerical input attribute. Given an answer set for a sub-space, to enforce equal selection probability, HDSampler will at most select a single data record as the sampled unit. The distinct aspect of the TPS algorithm is that it will use several of the obtained data records, but then would modify the estimators to account for unequal selection probabilities. In this way, with the same sampling cost, TPS could have more sampled units than HDSampler. In other words, TPS can have a much lower sampling cost for a fixed sample size.

The algorithm proceeds as follows.

1. Suppose there are a total of \( M \) sub-spaces in \( D \). We randomly choose \( m \) sub-spaces. This can be done by random assigning valid values or ranges for the input attributes and issuing the corresponding deep web queries to obtain the answer set. The sampling cost for this stage can be viewed as being proportional to \( m \).

2. For the \( i^{th} \) sampled sub-space, we select a first-phase random sample of size \( n_{i1} \). Since the data records in the \( i^{th} \) sampled sub-space have been delivered to the system, selecting these samples does not cause any additional queries, and thus can be viewed as free of sampling costs. One exception may be that we need to click the
“next” button in the web browser to obtain additional page(s), but we consider this a part of the original sampling cost.

3. For the \( i^{th} \) sampled sub-space, if any unit in the first-phase sample satisfies the limit condition, we select a second-phase random sample from the sub-space with the size \( n_{i2} \). Otherwise, we do not select the second-phase sample, i.e., \( n_{i2} = 0 \). These are also obtained from the data records obtained originally, and thus, we view second-phase sampling as not imposing any additional costs. The underlying idea is that based on neighborhood proximity, if any unit in the first-phase sample is a unit of interest, it is likely that this sub-space contain other units of interest.

Then, for the \( i^{th} \) sampled sub-space, the final sample is \( s_i \) with the size \( n_i = n_{i1} + n_{i2} \). We denote \( s_i = \{y_1, y_2, \ldots, y_{n_i}\} \). We use \( N_i \) to denote the total number of data records in the \( i^{th} \) sub-space, which is the total number of matched records of a query that most data sources often return. We first try to find estimators for AVG and SUM for the \( i^{th} \) sampled sub-space.

**Estimator for SUM for the \( i^{th} \) Sub-space:** Since selection probability for the units in the TPS sample for the \( i^{th} \) sub-space is not uniform, an intuitive estimator for SUM, \( \hat{t}_i = \frac{N_i}{n_i} \sum_{j=1}^{n_i} y_j \), is biased. In the following, we propose a unbiased estimator for \( \hat{t}_i \).

In the following, we first propose a trivial unbiased estimator then, we use the **Rao-Blackwell Theorem** to improve the efficiency of the trivial unbiased estimator.

Considering the first two drawn units \( y_{i1} \) and \( y_{i2} \) in the TPS sample for the \( i^{th} \) sub-space only, we have a trivial unbiased estimator for the sub-space SUM as.

\[
\hat{t}_i = y_{i1} + \frac{y_{i2}(1 - p_{i1})}{p_{i2}}
\]

where \( p_{i1} \) and \( p_{i2} \) are the probabilities of choosing \( y_{i1} \) and \( y_{i2} \) in the sample for the \( i^{th} \) sub-space respectively.
We use two indicator variables, $I_{ij}$, which is 1 if $y_{ij}$ is the first sampled unit in the sample $s_i$ and 0 otherwise, and $I'_{ij'}$, which is 1 if $y_{ij'}$ is the second sampled unit in the sample $s_i$ and 0 otherwise. Then, we can write the above estimator as

$$\hat{t}_i = \sum_{j=1}^{N_i} \sum_{j'>j} y_{ij} I_{ij} + \frac{y_{ij'}(1 - p_{ij'})}{p_{ij'}} I'_{ij'}$$

Next, we improve the above estimator using the Rao-Blackwell Theorem.

**Theorem 3.3.4.** (Rao-Blackwell Theorem [111]) Let $T$ be any unbiased estimator of a parameter $\phi$, and let $W$ be sufficient for $\phi$. Define

$$T_w = E[T|W] = \eta(W)$$

Then $T_w$ is an unbiased estimator of $T$, and $T_w$ is more efficient than $T$.

Based on the Rao-Blackwell theorem, we improve the estimator $\hat{t}_i$. We define $s_i$ to be the final sample we selected for sub-space $i$ and $s_i$ is a sufficient statistic for $\hat{t}_i$. Applying the Rao-Blackwell theorem, we have a new estimator

$$\hat{t}_i = E[\hat{t}_i|s_i] = \sum_{j} \sum_{j'>j} (y_{ij} + \frac{y_{ij'}(1 - p)}{p}) \frac{P(s_i, t_i)}{P(s_i)}$$

$$= \sum_{j} \sum_{j'>j} \frac{y_{ij}P(s_i, t_i)}{P(s_i)} + \frac{y_{ij'}(1 - p)P(s_i, t_i)}{pP(s_i)}$$

We have

$$P(s_i, t_i) = P(s_i|t_i)P(t_i) = P(s_i|j, j')p \frac{p}{1 - p}$$

As a result, we have

$$\hat{t}_i = \sum_{j} \sum_{j'>j} \frac{P(s_i|j, j')}{P(s_i)} \left( \frac{py_{ij}}{1 - p} + py_{ij'} \right)$$

We denote $q_i$ to be the total number of units in the sample $s_i$ for the $i^{th}$ sub-space, which are the units of interest, and we have the following result for $\frac{P(s_i|j, j')}{P(s_i)}$ as shown
\[
\frac{P(s_i|j,j')}{P(s_i)} = \begin{cases}
\frac{N_i(N_i-1)}{n_i(n_i-1)} & n_{i2} = 0, \\
\frac{N_i(N_i-1)}{(n_i+n_{i2})(n_i+n_{i2}-1)} & n_{i2} > 0 \text{ and } q_i = n_{i2}, \\
\frac{N_i(N_i-1)(n_i+n_{i2}-2)!}{(n_i+n_{i2})!-n_{i2}!\frac{(n_i+n_{i2}+q_i)!}{(n_{i2}+q_i)!}} & n_{i2} > 0 \text{ and } q_i < n_{i2} \text{ and } nc(j,j'), \\
\frac{N_i(N_i-1)((n_i+n_{i2}-2)!-n_{i2}!\frac{(n_i+n_{i2}+q_i)!}{(n_{i2}+q_i)!})}{(n_i+n_{i2})!-n_{i2}!\frac{(n_i+n_{i2}+q_i)!}{(n_{i2}+q_i)!}} & n_{i2} > 0 \text{ and } q_i < n_{i2} \text{ and } nc(j,j').
\end{cases}
\]

Figure 3.1: \( \frac{P(s_i|j,j')}{P(s_i)} \) factor for the TPS algorithm

in Figure 3.1. \( nc(j,j') \) means that both \( j \) and \( j' \) satisfy neighborhood condition, and \( nc(j,j') \) means that neither \( j \) nor \( j' \) satisfies neighborhood condition.

**Estimator for SUM for Data Source \( D \):** Given the SUM estimator for the \( i^{th} \) sub-space, the SUM estimator for data source \( D \) using TPS is computed as follows

\[
\hat{t}_{TPS} = \sum_{i=1}^{M} \frac{\hat{t}_i}{\pi_i}
\]

where \( M \) is the total number of sub-spaces and \( \pi_i \) is the probability of choose a sub-space. Here, we have \( \pi_i = \frac{m}{M} \).

**Estimator for AVG for Data Source \( D \):** Similarly, the AVG estimator for \( D \) using TPS is as follows.

\[
\bar{y}_{TPS} = \frac{\hat{t}_{TPS}}{N}
\]
3.3.5 Estimating MAX and MIN

Using any of the above proposed algorithm, we obtain a sample of size \( n \) as \( S = \{r_1, r_2, \ldots, r_n\} \). The attribute of interest of \( r_i \) is denoted as \( y_i \). Then the estimator for the population maximum \( \hat{y}_{max} \), and population minimum \( \hat{y}_{min} \) are \( \hat{y}_{max} = \text{Max} \{y_i\} \) , and \( \hat{y}_{max} = \text{Min} \{y_i\} \).

**Theorem 3.3.5.** (Chebyshev Inequality [111]) If a random variable \( X \) has a finite mean \( \mu \) and a finite variance \( \sigma^2 \), then for any \( \epsilon \geq 0 \), we have \( \Pr[|X - \mu| \geq \epsilon] \leq \frac{\epsilon^2}{\sigma^2} \).

We consider the attribute of interest as a random variable \( X \), using Theorem 3.3.5, for MAX estimation \( \hat{y}_{max} \), we could find a bound on the probability that \( |X - \mu| \geq \hat{y}_{max} - \mu \). In Theorem 3.3.5, we set \( \epsilon = \hat{y}_{max} - \mu \), we could get \( \Pr[X \geq \hat{y}_{max}] \leq \frac{\sigma^2}{(\hat{y}_{max} - \mu)^2} \). In other words, we could say that \( \hat{y}_{max} \) is at least the \( 1 - \frac{\sigma^2}{(\hat{y}_{max} - \mu)^2} \) quantile of the values of the attribute of interest. We could build similar bound for the minimum estimator as well, which is omitted due to lack of space.

3.4 Extension

The algorithms proposed in Section 3.3 are used for aggregation queries with one aggregation attribute on a single data source. In this section, we provide some solutions to extend our methods to the cases of aggregation query with multiple correlated aggregation attributes and/or on multiple data sources.

3.4.1 Multiple Data Sources

Let us consider the following scenario. There are multiple data sources providing the data on a certain attribute but with different distribution. For example, We want to know the average sales for US companies in the last year. Suppose the US company
sales data is located in four data sources corresponding to the company headquarters located in the East, South, North and West part of the US respectively.

In this case, we could consider each data source as a stratum of the entire data, and we use the idea of stratified sampling to estimate the aggregation function on the attribute of interest. For example, if we want to estimate the average sales for US companies as in the above example, we know the size of the $i^{th}$ data source is $N_i$, and the total size of the entire data size is $N = \sum_{i=1}^{H} N_i$, where $H$ is the total number of data sources considered. Then, the stratified estimator is $\bar{y}_{str} = \sum_{i=1}^{n} w_i \times \bar{y}_i$, where $\bar{y}_i$ is the estimates from the $i^{th}$ data source, which can be obtained using the algorithms proposed in Section 3.3, and $w_i$ is the weight assigned to each stratum (data source), which can be computed as $w_i = \frac{N_i}{N}$.

### 3.4.2 Multiple Attributes of interest

Let us consider the following scenario. We have $k$ attributes of interest $\{A_{q1}, A_{q2}, \ldots, A_{qk}\}$. We use the same sampling algorithms as proposed in Section 3.3 to select the *initial samples.* Each selected units in the initial sample has $k$ attributes of interest need to be consider when determining whether the unit is a unit of interest. Recall that in the univariate case (one attribute of interest), if the value of the attribute of interest $y_i$ for the unit $s_i$ satisfies the limit condition $y_i > \eta$, $s_i$ is a unit of interest. However, in the multivariate case (multiple attributes of interest), the limit condition is specified by a *region* $H$ in $k$-dimensional space. For a unit $s_i$, if we use $y_{ij}$ to represent the value of the $j^{th}$ attribute of interest, the unit $s_i$ is a unit of interest if it satisfies the condition $y_1 \in H$, specifically, $y_{ij} > \eta_j$, where $\eta_j$ is the upper limit value for the $j^{th}$ attribute of interest.
The estimators for the univariate case are shown in Section 3.3, and these estimators is still valid to be used for the multivariate case even though the limit condition in the multivariate case for one sampled unit depends on multiple attributes [122].

3.5 Evaluation

In this section, we evaluate the sampling algorithms we have developed. First, we evaluate each of the sampling algorithms individually. Second, we compare our new sampling algorithms with HDSampler.

3.5.1 Data Sets

We have used 8 data sets, including six synthetic datasets and two real data sets.

**Synthetic Data sets:** The synthetic datasets were generated using MINITAB\(^2\), a statistical software package. We generated six datasets, corresponding to data skew values 0, 1, 3, 5, 7, and 9, and refer to them as synskew0, synskew1, synskew3, synskew5, synskew7, and synskew9, respectively. Each dataset has 1000 data records and 7 attributes (columns). The first attribute is the attribute of interest and the other 6 attributes are used as input attributes to query the data. Each of the 6 input attributes has a different neighborhood proximity error (NPE), detailed in Section 3.5.2, with respect to the attribute of interest. The NPEs for the 6 synthetic datasets range from 0.46 to 1.72, and larger NPE value indicates weaker neighborhood proximity.

**US Census Dataset:** The Census dataset comprises of the 2002 US Economic Census data on Wholesale Trade Product Lines listed by the Kind of Business. This

\(^2\)http://www.Minitab.com
dataset can be downloaded at American FactFinder\(^2\). This dataset is referred to as IBQ. After removing free text, we have 6 attributes and 24984 data records. We use the sales attribute as the attribute of interest, and the number of establishments as the input attribute. The skew of the sales data in IBQ is 8, and the NPE value is 0.67, implying that this dataset has good neighborhood proximity.

**Yahoo! Auto Data set:** The Yahoo! Auto data set, denoted as Auto, comprises of the data crawled from a subset of a real-world hidden database at http://autos.yahoo.com/. Particularly, we download the data on used Ford cars from any model between 2000 and 2009 and located within 50 miles of a zipcode address. This yields a dataset with 1146 data records. We consider the price attribute as the attribute of interest. The data skew of the price data in Auto is 0.7. The NPE value of Auto is 0.31, which also indicates good neighborhood proximity.

It should be noted that the data had to be downloaded from hidden databases, as in the case for Auto, because in order to evaluate the estimators, we must know the exact results for aggregation functions.

### 3.5.2 Evaluation Metrics

In our experiments, five metrics are used.

**Absolute Error Rate (AER):** AER, as defined in Section 3.2.1, captures the estimation accuracy, with a small AER value indicating higher accuracy. We show detailed results on AVG and MAX estimates as the results for SUM and MIN are similar.

**Direct Sample Size:** As stated in Section 3.3.1, the number of direct sampled units determines the sampling cost.

**(Projected) Sampling Cost:** The (projected) sampling cost measures the actual

\(^2\)http://factfinder.census.gov/
sampling cost of a sampling algorithm in terms of time. An initial study shows that
the average network transmission cost for the Yahoo! Auto web-site is about 250ms
for obtaining one direct sampled unit. We use this measurement to project the actual
sampling costs for our sampling algorithms.

**Total Sample Size:** Total sample size is the sum of the sizes of the direct sample
and the associated sample.

**Neighborhood Proximity Error:** To quantify neighborhood proximity for a partic-
ular data source, we define the metric *Neighborhood Proximity Error (NPE).* This
is computed by evaluating NPE of the attribute of interest with respect to the input
attributes. Let us first consider the NPE of the attribute of interest with respect to
the input attribute $A_i$ is denoted as $NPE_i$.

The computation is done as follows. We partition all records into clusters based
on the neighborhood of $A_i$. Suppose, we form $n$ clusters \{CL$_1$, CL$_2$, $\ldots$, CL$_n$\}. For
the cluster CL$_j$, we define $NPE_{ij}$ as

$$NPE_{ij} = \frac{\sum_{k=1}^{|CL_j|} |y_{jk} - \mu_j|}{|CL_j|}$$

where $y_{jk}$ is the $y$ value of the $k^{th}$ records in cluster CL$_j$, $\mu_j$ is the average value of
$y$ for all records in CL$_j$, and $|CL_j|$ is size of CL$_j$. Then, $NPE_i$ is defined as

$$NPE_i = \frac{\sum_{j=1}^{n} NPE_{ij}}{|A_i|}$$

where, $|A_i|$ is the number of neighborhoods formed for the attribute $A_i$.

Finally, we define the NPE of $y$ with respect to all input attributes $IN = \{A_1, A_2, \ldots, A_s\}$ in $D$ as

$$NPE = \frac{\sum_{i=1}^{s} NPE_i}{s}$$

The smaller the NPE value, the better is the neighborhood proximity.
3.5.3 Performance of HDSampler on Synthetic Skewed Datasets

In this experiment, we use HDSampler to select a SRS of size 50 (5% of total data records) to estimate the average and maximal values of the attribute of interest, repeating the experiment for each of the 6 synthetic datasets. We examine the AER of the estimates on data with different skews. From the results, we observe that with the increase of data skew, the accuracy of the estimation obtained using HDSampler degraded dramatically. For the data sets with skew of 3, 5, 7 and 9, the AERs of the AVG (MAX) estimates obtained by HDSampler are 40% (20%), 50% (30%), 70% (46%), and 88% (87%), respectively. This confirms that a random sampling approach like HDSampler is not appropriate for skewed data.

We conducted another experiment. Using the same datasets, we measure the least number of direct sampled units needed to achieve an AER of 10% using HDSampler. We observe that for the datasets with skew of 3, 5, 7 and 9, the necessary sample size needed by HDSampler is 22%, 50%, 70% and 90%, respectively, of the total dataset size.

3.5.4 Evaluation of ANS Algorithm

In this section, we evaluate the ANS algorithm, separately considering some variants, with different termination conditions and/or fixed/variable upper limit values.

Parameter Evaluation

For this experiment, we consider the cost limit to be infinite. This implies that only the first termination rule of ANS is used. In this case, there are two important parameters in the ANS algorithm, which are $k$, the number of units of interest in the initial sample, and the upper limit value $\eta$.

Impact of Parameters on AER: We first examine the effect of the two parameters
on the AER of the estimators. In this experiment, for each data set, we vary \( k \) from 2 to 30 and upper limit value \( \eta \) from 80% upper quantile of the data to 95% upper quantile. We record the AERs for the AVG and MAX estimates. The results for the datasets synskew3, synskew7, and IBQ are shown in Figure 3.2. Results for other data sets are similar and omitted due to lack of space. In the sub-figures in Figure 3.2, the x-axis is the \( k \) value and the y-axis shows the AER values.

From Figure 3.2, we have the following observations. For both AVG and MAX estimates, ARE decreases with the increase of \( k \). The greatest decrease in AER happens when \( k \) increases from 2 to 8, and for \( k \geq 8 \), either the decrease rate in AER slows down, or the AER stays still or even has a small increase. For AVG estimates, for different upper limit values and datasets with different skew, a relatively low AER of 20% can be achieved at \( k = 8 \). If we set upper limit quantile to be 0.95, and \( k = 8 \), we achieve an AER around 10% for the AVG estimates for all datasets. For MAX estimates, at \( k = 8 \), we achieve an AER around 15% for all datasets and upper limit quantile values. If we fix upper limit quantile to be 0.95, and \( k = 8 \), we can achieve an AER around 5% for MAX, across all data sets. This shows that while small \( k \) values impact the algorithm negatively, very large \( k \) values are not necessary. This is because for data with a large skew, small \( k \) cannot effectively find the data that causes the skew. But, once such data can be identified, accuracy is not improved by larger values of \( k \). Another observation is that when \( k \geq 8 \), AER of MAX estimation is lower than the AER of AVG estimation for most cases. The ANS algorithm can identify data with large values, and as MAX estimation depends only on the maximal value, the algorithm is more effective in estimating it. The results also show an expected result that for both AVG and MAX estimation, the AER values become lower with the increase of upper limit quantile values.

**Impact of Parameters on Sampling Costs:** In this experiment, we vary the value
of $k$ and the upper limit quantile. We record the direct sample size and (projected) sampling cost. The results for synskew3, synskew7 and IBQ are shown in Figure 3.3. In the sub-figures in Figure 3.3, the x-axis is the $k$ value, the left y-axis shows the direct sample size (bar chart), and the right y-axis shows the (projected) sampling cost in term of seconds (line chart).

From Figure 3.3, we observe that with the increase of $k$ and the upper limit quantile, both the direct sample size and sampling costs increase. For all upper limit quantile cases, the sampling costs for $k > 10$ exceed 40 seconds. Based on the results from the above two experiments, in our following experiments, with the ANS algorithm, we set $k = 8$ and upper limit quantile to be 0.95.

**Impact of Data Skew:** Besides $k$ and $\eta$, data skew is another factor which impacts the performance of the ANS algorithm. In this experiment, we show the AER values of the ANS algorithm on data with different skew. The results are shown in Figure 3.4. We observe that for both AVG and MAX estimation, AER increases moderately (i.e. the accuracy decreases) with the increase in data skew. When $k = 8$, which is the default $k$ value, the AERs of the ANS algorithm are always lower than 19% for different data skew which illustrates the effectiveness of our ANS algorithm.

**Evaluation of Bias**

In Section 3.3.3, Lemma 3.3.3 shows that the bias of the ANS estimator is very small if the $k$ value is not large, such as $k = 8$. In this experiment, we validate this aspect. We use the IBQ data set, and consider two cases, which correspond to cost limits of 200 and 500 direct sampled units, respectively. Since the bias analysis in Section 3.3.3 was performed for the AVG estimate, we focus on the AVG estimate in this experiment. For each case, we show the AERs of the AVG estimate and the direct sample sizes with respect to the different values of $k$. The results are shown in
Figure 3.5. The results for other datasets are similar. In Figure 3.5, the x-axis is the $k$ values, the y-axis on the left panel shows the AERs of the AVG estimate, and the y-axis on the right panel shows the direct sample sizes.

For the cost limit of 200, on the left panel in Figure 3.5, AER first decreases to about 7% when $k = 8$, then when $k > 10$, AER increases (bias occurs) moderately. From the corresponding chart of the direct sample size on the right panel in Figure 3.5, for the cost limit of 200, we observe that when $k = 10$, we reached the cost limit of 200. We have similar observation for cost limit of 500 case. AER begins to increase moderately when $k = 30$ (left panel), which corresponds to where the cost limit of 500 is reached (right panel). Figure 3.5 show that when the cost limit is reached, bias indeed occurs. However, as long as the $k$ value is relatively small w.r.t. the dataset size, the increased AER value is still reasonable, and we can still achieve good estimation using the ANS algorithm.

ANS with Post-stratification: Trade off between Sample Size and AER

In Section 3.3.2, we show that if a upper limit value $\eta$ brings too many neighbors into the sample, we increase $\eta$, resulting in a decreased sample size and lowered sampling cost. There is a trade-off between reducing sample size and estimation accuracy. In this experiment, we evaluate the ANS algorithm with dynamically adjusting $\eta$ values (post-stratification), using IBQ and synskew3. The results for other data sets are similar. By adjusting $\eta$ by different values, we control the percentage of the sampled units saved. Then, for each case, we record the difference of the AER values of the AVG estimate between using post-stratification and without post-stratification (i.e., using a fixed $\eta$). The results are shown in Figure 3.6.
From Figure 3.6, we observe that the differences of AER between using post-stratification and without post-stratification are always within 10% (very small decrease in estimation accuracy), even if we saved a relatively large (35%) percentage in sample size. This shows the effectiveness of our ANS algorithm with post-stratification. If we significantly reduce the sample size (over 50%), the estimation accuracy is lowered significantly, as we will expect.

3.5.5 Evaluation of TPS Algorithm

In this subsection, we evaluate the TPS algorithm.

Effect of Initial Selected Sub-space Size: The number of sub-spaces selected is important for the TPS algorithm. In this experiment, we evaluate the performance of TPS w.r.t. the number of sub-spaces selected. Intuitively, the larger the number of sub-spaces selected, the better the estimation accuracy. However, larger number of selected sub-spaces also suggests higher sampling cost. In this experiment, for each data set, we vary the percentage of selected sub-spaces w.r.t. the total number of sub-spaces, and record the corresponding AER values of the AVG and MAX estimates as well as the direct sample sizes. The results for synskew1, synskew3, IBQ and Auto are shown in Figure 3.7. The results for other data sets are similar. In the sub-figures in Figure 3.7, the x-axis is the percentage of selected sub-spaces, the left y-axis is the AER values and the right y-axis shows the direct sample sizes.

From Figure 3.7, we observe that for all data sets, the AER for both AVG and MAX estimates decreases (better accuracy) with the increase of selected sub-space size. However, the direct sample size increases with the increase of the selected sub-space size. The AERs for both AVG and MAX estimates are always smaller or around 15% for selected sub-space size of 30%. Furthermore, selected sub-space size of 30% also corresponds to small direct sample size for all data sets (below 1% of the total
data set size). This shows that with small sampling cost, the TPS algorithm could achieve good estimation accuracy. In our following experiments, without specification, we set the selected sub-space size to be 30%.

**Effect of Data Skew:** Here, we measure the AERs of the AVG and MAX estimates using TPS on synthetic datasets with different skew levels. The results are shown in Figure 3.8. Similar to the ANS algorithm, the AER of both AVG and MAX increases when there is a larger data skew. However, for our chosen selected sub-space size, which is 30%, the AERs are always smaller than 17%. This shows that TPS works well even when there is a significant skew.

### 3.5.6 Comparison of Three Sampling Algorithms

In this section, we compare the performance of our two new sampling algorithms, ANS and TPS, with HDSampler.

**Absolute Error Rate Comparison**

We first compare the estimation accuracy of the three sampling algorithms. We fix the direct sample size, and compare the AER values of both AVG and MAX estimates of the three algorithms. For each of the 6 synthetic datasets, we fixed direct sample size to be 5% of the total dataset size. The result are shown in Figures 3.9.

From Figure 3.9, we observe that for small skew (skew \( \leq 1 \)), the AERs of the three sampling algorithms are comparable. However, with the increase in data skew, the estimation accuracy of HDSampler degraded severely. In these cases, ANS and TPS algorithms outperform HDSampler, with AER being better by a factor of 4 on the average. In particular, the AER values of ANS and TPS are always below 20%. The performance of ANS and TPS are close to each other.

For data sets **IBQ** and **Auto**, since their data skew is fixed, we vary the direct sample
For each fixed direct sample size, we compare the AER values of the AVG and MAX estimates of the three algorithms. The results are shown in Figures 3.10 and 3.11.

For IBQ (Figure 3.10), for both MAX and AVG, the performance of ANS and TPS is better than HDSampler for all direct sample sizes, by a factor of 3 on the average. For Auto, ANS and TPS outperform HDSampler for MAX, with average AER being lower by a factor of 3. This shows the advantage of using ANS and TPS for obtaining the large value data, even when the skew is small. For AVG estimation on Auto, since the data skew is small, the performance of the three algorithms is comparable.

The reasons that ANS and TPS algorithms outperform HDSampler are as follows. First, ANS and TPS sample more skew causing data. Second, a large number of the sampled units in ANS and TPS algorithms are associated sampled units, which are free of sampling cost. Thus, compared with HDSampler, they also have low sampling costs. Table 2 shows the total sample size obtained by the three algorithms in the above experiment on IBQ data fixing sampling cost (the results on Auto is similar). We observe that with the same sampling cost, ANS and TPS sample more data so that their estimation accuracy is higher than HDSampler.

The Effect of Neighborhood Proximity: HDSampler vs ANS

In this experiment, we evaluate the effect of neighborhood proximity on estimation accuracy of the ANS algorithm. In each synthetic dataset, there are 6 input attributes, each of which has a different neighborhood proximity error (NPE) with respect to the attribute of interest. We execute ANS with cost limit of 100 direct sampled units on synthetic data sets using different input attributes and examine the AERs of the AVG and MAX estimates. The result on synskew5 is shown in Figure 3.12. The results on other synthetic datasets are similar. In Figure 3.12, the x-axis is the neighborhood
proximity errors (larger error means weaker neighborhood proximity), and the y-axis is the AERs. The largest NPE in Figure 3.12 indicates the input attribute that is generated completely independent from the attribute of interest.

Since HDSampler doesn’t explore neighboring units, we have horizontal lines for HDsampler. From Figure 3.12, we observe that with the increase of NPE, the estimation accuracy of ANS decreases. This is reasonable, because weaker neighborhood proximity implies that larger number of neighboring units selected are not skew causing data. However, the AERs of ANS, even for the largest NPE case, are consistently lower than HDSampler. This shows that even for data sets with weak neighborhood proximity, ANS still outperforms HDSampler.

**Sampling Cost Comparison between HDSampler and TPS for Small Skew Data**

In Section 3.5.6, we had shown that for data with a small skew, the estimation accuracy of HDSampler and TPS is comparable. In this experiment, we show that
for data with only a small skew, to achieve the same AER, TPS incurs lower sampling costs than HDSampler. We use two small skew data sets, synskew1 and Auto (skew=0.7). We vary the AERs, and record the least number of direct sampled units needed to achieve the AER for TPS and HDSampler. We also record the total sample size (direct sampled units and associated sampled units) for TPS. The results for two datasets are shown in Figure 3.13 and Figure 3.14.

Figure 3.13 and 3.14 show the results on synskew1 and Auto data respectively. The numbers around each data point in these Figures show the sampling cost in terms of number of seconds. We have the following observations. First, to achieve a low AER, such as AER= 2% or 5% (high accuracy), TPS requires significantly less direct sampled units and less sampling costs. These costs are about one third of that of HDSampler, on both the datasets. Second, although the sampling cost of TPS is lower than HDSampler, the total number of sampled units TPS uses are larger than HDSampler. This illustrates the key advantage underlying TPS.

**The Effect of Query Selectivity**

In this experiment, we vary the selectivity of queries from 10% to 100% and record the AERs for AVG and MAX estimates of the three algorithms. The result on IBQ data is shown in Figure 3.15. We have the following observations. First, for MAX estimation, the ANS and TPS algorithms outperform HDSampler for all query selectivity values by a factor of 3 on the average. Second, for AVG estimation, the ANS algorithm consistently outperforms HDSampler by a factor of 2. The TPS algorithm outperforms HDSampler for moderate and large selectivity queries, but is comparable with HDSampler for low selectivity queries. This is because many selected sub-spaces in TPS are filtered out if the query has a low selectivity.
Figure 3.2: The Effect of $k$ and $\eta$ on AER - ANS Algorithm: (a) synskew3, (b) synskew7, (c) IBQ (skew=8)
Figure 3.3: The Effect of $k$ and $\eta$ on Sampling Cost - ANS Algorithm: (a) synskew3; (b) synskew7; (c) IBQ (skew=8)

Figure 3.4: ANS Performance w.r.t. Data Skew, Upper Limit Quantile=0.95
Figure 3.5: Analysis of Bias for ANS on IBQ Data, Cost Limit=200/500

Figure 3.6: ANS Algorithm With Post-stratification
Figure 3.7: The Effect of Selected Sub-space Size - TPS Algorithm: (a) synskew1; (b) synskew3; (c) IBQ (skew=8); (d) Auto (skew=0.7)

Figure 3.8: TPS Performance w.r.t. Data Skew
Figure 3.9: Compare AER for Three Algorithms on Synthetic Data

Figure 3.10: Comparing AER for Three Algorithms on IBQ (skew=8)

Figure 3.11: Comparing AER for Three Algorithms on Auto (skew=0.7)
Figure 3.12: ANS Performance Evaluation w.r.t. Neighborhood Proximity on synskew5

Figure 3.13: Sampling Cost Comparison for TPS and HDSampler Fixing AER Level on synskew1

Figure 3.14: Sampling Cost Comparison for TPS and HDSampler Fixing AER Level on Auto(skew=0.7)
Figure 3.15: Three Algorithm Comparison: Varying Query Selectivity on IBQ
3.6 Summary

In this chapter, we have developed two novel adaptive sampling algorithms, adaptive neighborhood sampling (ANS) and sub-space based two phase adaptive sampling (TPS), to approximately answer deep web aggregation queries. Without any prior knowledge of hidden data's distribution, our algorithms can obtain accurate estimations with low sampling costs. Our detailed experimental evaluation has shown that 1) For data with a moderate or a large skew, ANS and TPS algorithms obtain an average estimation accuracy around 90% for AVG and 95% for MAX, which is about 4 times better than HDSampler. 2) For data has only a small skew, the TPS method has only one-third of the sampling costs of HDSampler. 3) We confirm our theoretical analysis that the bias of the ANS estimator does not impact the estimation performance, and 4) By adaptively adjusting the upper limit value \( \eta \) in the ANS algorithm, we can reduce the sampling costs by one-third, with less than 10% decrease in the estimator accuracy.
In this chapter, we study the problem of estimating the result of an aggregation query with low selectivity when a data source only supports limited data accesses. Existing stratified sampling techniques cannot be applied to such a problem since either it is very hard, if not impossible, to gather certain critical statistics from such a data source, or more importantly, the selective attribute of the query may not be queriable on the data source. In such cases, we need an effective mechanism to stratify the data and form homogeneous strata with respect to the selective attribute of the query, despite not being able to query the data source with the selective attribute.

This chapter presents and evaluates a stratification method for this problem utilizing a queriable auxiliary attribute. The breaking points for the stratification are computed based on a novel Bayesian Adaptive Harmony Search algorithm. This method derives from the existing Harmony search method, but includes novel objective function, and introduces a technique for dynamically adapting key parameters of this method. Our experiments show that the estimation accuracy achieved using our method is consistently higher than 95% even for 0.01% selectivity query, even when
there is only a low correlation between the auxiliary attribute and the selective attribute. Furthermore, our method achieves at least a five fold reduction in estimation error over three other methods, for the same sampling cost.

In Section 4.2, we give the background on stratification and the original harmony search algorithm. Section 4.3 gives an overview of our proposed approach. The details of our Bayesian adaptive harmony search stratification algorithm is presented in Section 4.4. Our technique is evaluated in Section 4.5.

4.1 Introduction

We study the problem of estimating the result of an aggregation query with low selectivity over a data source that only allows limited access to data, i.e., is limited with respect to the kind of queries it supports. By low selectivity, we refer to the queries with a selection predicate that is matched by only a very small fraction of the data. Data sources with limited access to data have been studied extensively in the context of data integration and query optimization [109, 31]. More recent examples of such data sources are the deep web data sources. With the rapid growth of the size and popularity of the data disseminated over the World Wide Web, a large portion of structured web data resides in the deep web. To access data in the deep web, a user must issue queries through the input interfaces of deep web data sources. Then, answers are returned to the user as dynamically generated HTML pages.

Deep web data sources often have limited data accessibility, from two different aspects. First, it is very hard, if not impossible, to obtain certain critical population statistics on deep web data. This is because many deep web data sources limit the number of queries a particular IP address can issue, or the number of data records can be returned, to protect their data from being completely downloaded, or to disallow a denial-of-service attack. Second, more importantly, deep web data can
only be accessed through query interface made available on the data source’s web
text. Therefore, certain attribute of a query may not be directly queriable over the
input interface of the deep web data source.

In this paper, we propose solutions to make online aggregation [66] feasible even
for low selectivity queries over deep web data sources with limited data accessibility.
Answering database queries by sampling has been extensively studied in the litera-
ture [67, 102, 30, 28, 29], including recent work on sampling methods and estimators
for low selectivity queries [78]. However, these approaches are not directly applica-
table to a database that offers limited data access. Similarly, sampling deep web data
sources has been a topic of recent interest [41, 39, 40]. However, sampling for low
selectivity queries has not been studied in this work.

To further illustrate our problem, let us consider the following example. Suppose
we have a data set as shown in Table 4.1. In this data set, we have 3 attributes and
10 records. We are considering the query Q1, which is

```
SELECT AVG(C) FROM Table 1 WHERE B>15
```

The selection predicate of Q1 only accepts data records from Table 4.1 with the
value of attribute $B$ larger than 15. From Table 4.1, we observe that there are

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1</td>
<td>58</td>
<td>32</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>60</td>
<td>2</td>
<td>79</td>
<td>51</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>59</td>
<td>3</td>
<td>63</td>
<td>70</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>52</td>
<td>2</td>
<td>100</td>
<td>24</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>2</td>
<td>17</td>
<td>100</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 4.1: A Low Selectivity Query Example
only 2 records that can be accepted by the selection predicate of $Q1$. The natural way of preparing to handle such low selectivity query is to use stratification [111]. Specifically, we can partition the entire data set into multiple sub-groups, called \textit{strata}, such that the values of the selective attribute(s) of the query (attribute $B$ in Table 4.1) are homogenous within each stratum. This objective is best achieved when the variability of the \textit{partitioning attribute(s)}, in our case the selective attribute(s), is minimized within each stratum, and maximized across different strata. For answering the query shown above, we could have two strata, i.e., the last two records (in bold) in one stratum, and the rest eight records in the other. After stratification, we know that the query’s selection predicate will favor certain strata. Then, those favored strata can be heavily sampled so as to obtain a better estimation for the query. More specifically, if a specified number of samples, $n$, are to be drawn from the data set, the number of samples selected from each stratum can be determined by existing sampling allocation methods, such as \textit{Neyman allocation} [111] or \textit{Bayes-Neyman allocation} [78].

Therefore, the critical problem is to find an \textit{appropriate stratification}, i.e., to find multiple \textit{breaking points} of the partitioning attribute on which the stratification is performed. In the literature, various methods have been proposed for this purpose, including \textit{clustering based stratification} [111] and \textit{outlier indexing} [28]. The above methods require knowledge of either the distribution or summary statistics of the partitioning attribute. While this information can be easily obtained for a relational database, this is not the case for a deep web source where the partitioning attribute is not directly queriable.

In statistical sampling, if the partitioning attribute is unknown or difficult to obtain, stratification can be performed based on an \textit{auxiliary} attribute, which is \textit{closely correlated} with the partitioning attribute [35]. Examples of methods which use a closely correlated auxiliary attribute include Dalenius and Hodges’s method [36,
However, all of the above methods have rigid requirements on the auxiliary attribute. For example, the auxiliary attribute and the partitioning attributes should be highly correlated and the variance of the auxiliary attribute is equal to the variance of the partitioning attribute [35], or the auxiliary attribute should be uniformly distributed within each stratum [49, 57]. In actual deep web data sources, these requirements are rarely satisfied. In our evaluation (Section 4.5), we actually show that the above methods do not achieve good estimation results for low selectivity queries on real data sets.

4.1.1 Problem Formulation and Our Contributions

In this paper, we have developed a Bayesian Adaptive Harmony Search Stratification algorithm for a non-queriable selective attribute based on an auxiliary attribute. We consider aggregation queries over deep web sources in the following format:

\[
\text{SELECT AVG}(f_1(r:Target)) \text{ FROM R As } r \text{ WHERE } f_2(r:Sel)
\]

Here, \(f_1()\) is any numerical function, \(f_2()\) is a selection predicate with low selectivity, and \(r:Target\) is the target attribute on which the aggregation is performed. If \(r:Sel\) is directly queriable on the input query form of the deep web data source, the query can be answered trivially by specifying the selection predicate directly on the input query form. However, when \(r:Sel\) is not directly queriable on the input query form, because of the limited query interface of a deep web data source, we require novel techniques.

A specific motivating example from the biology domain is as follows. Suppose we are interested in Single Nucleotide Polymorphisms (SNPs), which are promising for explaining the genetic contribution to certain rare diseases [2]. Biologists want to
obtain the average frequency of the SNPs related with a certain type of rare breast cancer. In this example, the target attribute is SNP frequency and the selective attribute is SNP function. SeattleSNP\(^1\) is a widely used deep web data source for searching SNP related information. The input interface of SeattleSNP is shown in Figure 4.1.

As we can see from this figure, the SNP data can only be queried by SNP identifier and the chromosome range of a SNP. This limited query interfaces prohibits us from directly querying the selective attribute SNP function and performing stratification.

In our proposed approach, we take certain directly queriable attribute on the input query form, the input attribute \(r.\text{Input}\), as the auxiliary attribute. We have developed an optimization to the recently proposed meta-heuristic algorithm, the harmony search algorithm [91], that we refer to as the Bayesian Adaptive Harmony Search. This algorithm finds robust stratification plans for the non-queriable selective attribute based on the auxiliary attribute. The stratification thus obtained can accurately reflects the distribution of the hidden selective attribute. In the above motivating example, we could use chromosome range as the auxiliary attribute to do stratification since there will be some correlation between the position of the chromosome on which a SNP is located and the function of the SNP.

\(^1\)http://pga.gs.washington.edu/
Overall, our contributions are summarized as follows.

1. We have developed a novel stratification method to stratify a hidden selective attribute based on an auxiliary attribute. The stratification obtained from our algorithm accurately reflects the distribution of the hidden selective attribute even when the correlation between the auxiliary attribute and the selective attribute is weak.

2. Besides applying the idea of Harmony search to the stratification for low selectivity queries on the deep web, an important contribution of our work is an approach for automatically adapting the values of the parameters in the original harmony search algorithm based on the bayesian approach.

3. We experimentally show that the estimation accuracy, obtained with the same sampling cost, based on the stratification generated from our method is better than three existing methods by at least a factor of 5. Put another way, we incur much lower sampling cost to obtain the same level of accuracy. Furthermore, the estimation accuracy we obtained is consistently higher than 95% even for extremely low selectivity query (0.01% selectivity).

4.2 Background

4.2.1 Stratification

In survey sampling, if the attribute we are interested in takes on different mean values in different sub-populations, we may be able to obtain more precise estimates to answer a given query by taking a stratified sample [111]. Stratified sampling is widely used for answering low selectivity queries [30, 28, 29, 78].

The first step in stratified sampling is stratification, i.e., to partition the data set $R$ into $k$ disjoint strata, with $R_i$ representing the $i^{th}$ stratum. Then, we have $R = R_1 \cup R_2 \cup \ldots \cup R_k$. The attribute on which stratification is performed is referred
to as the *partitioning attribute*. Specifically, suppose the partitioning attribute $x$ has the range $[x_0, x_k]$, then we need to find $k - 1$ breaking points within the range $[x_0, x_k]$, such that $x_0 < x_1 < x_2 < \ldots < x_{k-1} < x_k$. In this way, the data records whose attribute $x$ fall into the range $[x_{i-1}, x_i]$ are in stratum $R_i$. Formally, we could consider a stratification of the partitioning attribute to be a vector of breaking points, for example, $\text{Vector}(x) = [x_0, x_1, \ldots, x_k]$.

As a specific example, we consider a dataset on sales of used cars. Suppose we are interested in estimating the average price of used cars with mileage less than 50000. We could stratify the data set on the attribute *Mileage*. A possible stratification could be as follows: $[0, 10000]$, $[10000, 40000]$, $[40000, 100000]$, and $[100000, 300000]$. In this example, the breaking points are mileage 10000, 40000, and 100000.

When stratification is performed, the size of the stratified sample we need to choose from each stratum $R_i$ can be determined by a sample allocation method, which will need to meet the constraint $\sum n_i = n$, where $n_i$ is the number of samples we draw from stratum $R_i$, and $n$ is a pre-defined total number of samples.

In sampling a hidden data source, i.e., a data source supporting only a limited access, for a low selectively query, we also need to determine the breaking points of the partitioning attribute for stratification. The key challenge we face is that the *selective attribute* of the query we are interested in, may not be directly queriable. As a result, stratification problem becomes significantly harder. The only option is to stratify or partition attribute with a queriable or an *auxiliary attribute*. Returning to the used car sale example, *target attribute* we are interested in is *Price*, and the the selective attribute is *Mileage*. Suppose *Mileage* cannot be queried on, and the input query form of the data source only provides direct access to the attribute *Year of Car*. Therefore, we need to take *Year of Car* as the *auxiliary attribute*, and create strata by choosing break points on these values. In choosing these break points, the
key consideration is to be able to efficiently find sufficient number of *useful samples*, i.e. samples that meet the selection predicate based on *Mileage*, for the given low selectivity query.

### 4.2.2 Harmony Search

The harmony search (HS) algorithm is a meta-heuristic algorithm that was developed in 2005 [91]. Harmony search is a phenomenon-mimicking algorithm inspired by the improvisation process of musicians. In the HS algorithm, each musician (decision variable) plays (generates) a note (a value) for finding the best harmony (global optimum). Harmony search algorithm had been very successful in a wide variety of optimization problems [53, 91], and offers several advantages with respect to traditional optimization techniques [100].

Harmony search algorithm tries to find a vector $X$ that optimizes (minimizes or maximizes) a certain objective function. The HS algorithm consists of the following five steps [91].

*Step 1: Initialize the optimization problem and algorithm parameters.* The optimization problem is specified as

$$
\text{Minimize } f(x_1, x_2, \ldots, x_N), \ x_i \in X_i, \ i = 1, \ldots, N
$$

where $f(X)$ is the objective function; $x_i$ is a decision variable in the vector $X$, $X_i$ is the set of possible values (range) for each decision variable $x_i$, and $N$ is the number of decision variables. To optimize the objective function, we need to specify the following harmony search parameters: the harmony memory size (the number of solution vectors in the harmony memory), harmony memory consideration rate ($HMCR$) and pitch adjusting rate ($PAR$). The $HMCR$ and $PAR$ parameters both range from 0 to 1 and determine the generation of a new harmony vector. $HMCR$ indicates our confidence on the existing solutions in the current harmony memory. If $HMCR$ is
set to be close to 1, it shows that we have high confidence that the existing solutions are close to the optimal one. However, if $HMCR$ is set to be close to 0, it means that we have low confidence on the existing solutions, and we are intend to obtain new solutions outside the current harmony memory. $PAR$ indicates the likelihood we assign a random shift to the value of a decision variable. A large $PAR$ indicates that we try to aggressively change the value of a decision variable, and a small $PAR$ suggests otherwise.

**Step 2: Initialize the harmony memory:** The harmony memory $HM$ is a $M \times N$ matrix, where $N$ is the number of decision variables and $M$ is the size of the harmony memory. In other words, the harmony memory contains $M$ randomly initialized solution vectors of the objective function. An example of initialized harmony memory is shown as follows.

$$HM = \begin{bmatrix}
    x_1^1 & x_2^1 & \ldots & x_{N-1}^1 & x_N^1 \\
    x_1^2 & x_2^2 & \ldots & x_{N-1}^2 & x_N^2 \\
    \vdots & \vdots & \vdots & \vdots & \vdots \\
    x_1^{M-1} & x_2^{M-1} & \ldots & x_{N-1}^{M-1} & x_N^{M-1} \\
    x_1^M & x_2^M & \ldots & x_{N-1}^M & x_N^M
\end{bmatrix}$$

**Step 3: Improvise a new harmony from the harmony memory:** A new harmony vector, $x' = [x_1', x_2', \ldots, x_N']$ is generated in a randomized fashion based on the value of the $HMCR$ and $PAR$ parameters. Specifically, the value of the design variable $x_i'$ can be chosen in two steps.

In the first step, we consider the harmony memory consideration rate, i.e., we choose an existing harmony or we generate a new harmony from an appropriate domain. Suppose we have a biased coin with head probability of $HMCR$, the value of $x_i'$ is generated based on either of the following two outcomes of tossing the coin.
If the head of the coin is up, the value of $x'_i$ should be chosen from any value of the $i^{th}$ decision variable in the current harmony memory. If the tail of the coin is up, the value of $x'_i$ should be chosen from any possible value within the range of the $i^{th}$ decision variable, which is $X_i$.

In the second step, for each $x'_i$, we determine whether it should be pitch-adjusted by a random distance by considering the $PAR$ parameter. Similarly, the value of $x'_i$ can be either shifted by a random distance with the probability $PAR$, (i.e. not shifted with the probability $1 - PAR$).

Overall, the rule for generating a new value for $x'_i$ is summarized as follows ($w/p$ stands for with probability).

$$x'_i = \begin{cases} 
(x'_i \in \{x^1_i, \ldots, x^M_i\}) + \delta & w/p \ HMCR \times PAR, \\
(x'_i \in X_i) + \delta & w/p \ (1 - HMCR) \times PAR, \\
(x'_i \in \{x^1_i, \ldots, x^M_i\}) & w/p \ HMCR \times (1 - PAR), \\
(x'_i \in X_i) & w/p \ (1 - HMCR) \times (1 - PAR)
\end{cases}$$

**Step 4: Update harmony memory:** If the new harmony vector is better than the worst harmony vector in the current harmony memory (in terms of the objective function value), the worst harmony vector in the harmony memory is replaced by the new harmony vector.

**Step 5: Termination:** The above iterative procedure is terminated until a termination condition is satisfied, for example, when the total number of iterations exceeds a threshold or when we achieve convergence in the objective function.

Our proposed Bayesian adaptive harmony search stratification algorithm derives from Harmony search. However, we had to address the following challenges. First, we need to map the stratification problem on a hidden data source to the harmony search model. Second, we had to develop an objective function that would be appropriate
for low selectivity queries. Third, since execution time is critical while executing a query, we developed a Bayesian framework to dynamically update \( HMCR \) and \( PAR \) parameters, and help the search converge faster. Details of these are presented in the next 2 sections.

### 4.3 Algorithm Overview

In this section, we give an overview of our entire algorithm. Our approach comprises of the following three steps. First, we map the stratification problem to the harmony search model. Second, we apply the harmony search algorithm to find the best stratification. To enable this step, we have developed a novel Bayesian adaptive framework to adjust the two parameters in the original harmony search algorithm, which are the harmony memory consideration rate and the pitch adjusting rate. The Bayesian adaptation method is detailed in Section 4.4. Finally, given the stratification obtained from the previous step, we apply a sample allocation method we have developed to allocate samples into each stratum.

#### 4.3.1 Modeling Harmony Memory

To estimate the answer of a low selectivity query accurately, we need to find the best stratification of the selective attribute, from the viewpoint of most accurately reflecting its distribution and being able to efficiently obtain sufficient number of samples that match the selection predicate. Let us suppose a selective attribute has a distribution as illustrated in Figure 4.2(a). A good stratification of this selective attribute is illustrated by the vertical reference lines in the Figure 4.2(a). Suppose a low selectivity query asks for data with the values in the range between 1.2 and 1.8. Using the shown stratification, a sample allocation method could obtain good estimation result by heavily drawing samples from the last stratum, \(([1.2, 2.1])\).
As we have described earlier, our focus is on cases where the selective attribute is not accessible through the input interface of the deep web data source. In such a case, we use an attribute available in the input query form as the auxiliary attribute. Unlike many existing methods in statistics, we cannot assume that there be a strong correlation between the auxiliary attribute and the selective attribute, since this assumption is not likely to hold on real data sources. In Figure 4.2(b), the solid line shows the distribution of the selective attribute, and the dotted line shows the distribution of an auxiliary attribute. In this example, the correlation between them is 0.35. By applying our approach, we are trying to find a stratification of the auxiliary attribute (shown as the vertical lines in Figure 4.2(b)), which is sufficiently close to the stratification we could obtain for the selective attribute.

Harmony search can be viewed as a guided search process, which can stratify the auxiliary attribute in a way that we can efficiently obtain samples meeting the selection predicate, and accurately estimate the aggregation function. In Section 4.2, we had described that the harmony memory consists of a list of decision vectors. In our specific problem scenario, we consider each stratification of the auxiliary attribute as a decision vector. In other words, each decision variable in our case is a breaking point value in the range of the auxiliary attribute. For example, the stratification shown in Figure 4.2(b), [0.25, 1.05, 1.25], is a decision vector in our harmony memory. In statistics literature, Cochran [33] concluded that in stratified sampling, there is typically not much further reduction in variance from having more than 6 strata. Based on this observation, in our implementation, the size of the decision vector is set to be 6, though clearly, our approach can apply in other cases as well.

At the beginning, the initial harmony memory is randomly generated. Suppose there are $M$ decision vectors in the harmony memory, i.e., the size of the harmony memory is $M$. Let the range of the auxiliary attribute be $X$. Then, each decision
Figure 4.2: Example: Auxiliary Attribute Based Stratification: (a) Stratification for Selective Attribute, (b) Auxiliary Attribute Based Stratification

vector is generated in the following three steps. First, we generate a random number, denoted as $k$, between 1 and 6 using a uniform distribution. This is the number of strata represented by this decision vector. Second, we need to generate $k - 1$ breaking points. Therefore, we generate $k - 1$ random numbers within the range of the auxiliary attribute, which is $X$, using a uniform distribution. Each such generated number is a breaking point. Third, we form a decision vector of size 6 by ranking the $k - 1$ breaking points in an increasing order and randomly inserting $6 - k$ number of $\ast$. The $\ast$ indicates an empty position in the stratification.

The above procedure is repeated $M$ times to initialize the harmony memory. An example of a harmony memory in our approach is shown as follows.
The above harmony memory shows 5 stratification candidates (decision vectors).
The first decision vector corresponds to the stratification of \([\text{Min}(X), 0.2], [0.2, 0.5], [0.5, 0.8], [0.8, \text{Max}(X)]\), where \(X\) is the range of the auxiliary attribute.

4.3.2 Harmony Search and Objective Function

Given the initial harmony memory, our stratification search algorithm works as follows.

1. For each initial stratification candidate in the harmony memory, compute its cost score. The method for computing this metric will be described later. A stratification candidate with a lower cost score is preferred.

2. Generate a new stratification candidate according to the original harmony search algorithm described in Section 4.2.2. However, a key distinctive aspect of our work is that in the search algorithm, the two harmony search parameters, \(HMCR\) and \(PAR\), are adapted according to our Bayesian framework. This method will be described later in Section 4.4.

3. For each newly generated stratification candidate, compute its cost score, and update harmony memory using the newly generated stratification candidate.

4. Repeat the above procedure until either of the following two termination conditions is satisfied: 1). the number of iteration exceeds a pre-defined threshold, or 2). the
average accuracy of the query obtained using the best stratification candidate in the current harmony memory is lower than a pre-defined threshold.

The method listed above clearly depends on the calculation of a cost or \textit{objective function}, which evaluates the quality of the stratification. We now describe our method for evaluating the quality of stratification. More specifically, given a stratification vector \( x = [x_1, x_2, \ldots, x_k] \) on an auxiliary attribute, our goal is to calculate the fitness of the stratification with respect to the distribution of the selective attribute.

In the traditional statistical methods, one prefers a stratification vector if the selective attribute of the data records in each stratum is homogenous. To test this condition, we could draw a random pilot sample of equal size from each stratum of the stratification, and compute the summation of the sample variance of the selective attribute of the pilot sample.

However, only considering the sum of variance condition is not sufficient for justifying a good stratification in our scenario, since the auxiliary attribute may not be highly correlated with the selective attribute. Thus, only using sum of variance may not ensure that we can efficiently obtain samples where the selection predicate will be true. Instead, we need to consider another measure in the objective function, which we call the \textit{precision} of a stratification. The \textit{precision} of a stratification is defined as the \textit{percentage} of the pilot sample (from current stratification) that satisfies the low selective predicate specified in the given query. The \textit{precision} of the \( i^{th} \) stratum in a stratification is formally defined as \( \text{pre}_i = \frac{|\{s_{ij} | f_2(s_{ij}) = \text{true}, s_{ij} \in S_i\}|}{|S_i|} \), where \( S_i \) is a simple random sample from the \( i^{th} \) stratum. Recall that \( f_2 \) is the selection predicate in the query. Stratification that involves a higher precision can help us in finding more data records satisfying the low selectivity predicate.

Formally, given a stratification \( x \), we draw a random pilot sample from each stratum \( R_i \), and compute two measurements. The first one is \( \sigma_i^2 \), which is the sample
variance, and the second one is $pre_i$, which is the precision. Then, the objective function for the stratification $x$ is defined as

$$f(x) = \sum_{i=1}^{k} \frac{\sigma_i^2}{pre_i}$$

Clearly, the above approach involves a significant sampling overhead during stratification itself. However, the stratification obtained allows us to obtain samples matching the selection predicate more effectively. Our experiments show that despite this overhead, we can obtain the same number of samples matching the selection predicate with a lower sampling cost.

4.3.3 Sample Allocation

The problem of sample allocation is to determine the number of samples $n_i$ that need to be drawn from each stratum $R_i$, so as to conduct the final estimation for the query. Suppose the total number of samples used to answer a query is pre-defined as $n$. Then sample allocation must satisfy the constraint $\sum_{i=1}^{k} n_i = n$. A widely used sample allocation method is Neyman Allocation [111]. The Neyman allocation states that the variance of the estimator is minimized when sample size $n_i$ is proportional to the size of the stratum, $N_i$, and to the sample variance of the selective attribute in the stratum, $\sigma_i^2$. That is,

$$n_i = \frac{n}{\sum_{j=1}^{k} N_j \times \sigma_j^2} N_i \times \sigma_i^2$$

We can see that Neyman allocation only captures one measurement of our stratification objective function, the sample variance, but not the second measurement, the precision. Based on the description in Section 4.3.2, a stratum should be sampled more heavily if one of the following two conditions hold: 1) The values of the selective attribute in the stratum is more heterogenous, i.e., the sample variance of
the selective attribute from the stratum is larger, and 2) The stratum covers more data records that satisfy the query’s selective predicate, i.e., the stratum has a higher precision.

To capture the above two measurements, we propose a novel sample allocation method. In our method, the sample size $n_i$ is proportional to the sample variance of the selective attribute in the stratum, and as well as the precision of the stratum. Formally,

$$n_i = \frac{n}{\sum_{j=1}^{k} \sigma_j^2 \times \text{pr} e_j} \times \sigma_i^2 \times \text{pr} e_i$$

### 4.4 Bayesian Adaptation For Harmony Search

From the description of harmony search in Section 4.2, we know that two parameters are very important for the performance of the harmony search algorithm, since they enable new, probably better, harmony vectors into the harmony memory. These two parameters are *harmony memory consideration rate* (*HMCR*) and *pitch adjusting rate* (*PAR*). As we explained earlier, *HMCR* indicates the probability that a new harmony vector is generated using the current solution candidates in the harmony memory, and *PAR* indicates the probability that we shift the values of a newly generated harmony vector by a random distance.

In the original harmony search algorithm [91], these two parameters are fixed during the entire process, though their initial values are chosen carefully. In applying harmony search idea to query processing, efficiency (rate of convergence) is a critical issue. We observe that the values of these two parameters should be dynamically adapted to achieve faster convergence. Specifically, at the beginning of the algorithm, we want to bring in more new harmony vectors into the harmony memory, as the randomly generated initial harmony vectors are less likely to be good. Thus, a small
value of \( HMCR \) and a high value of \( PAR \) is preferred. As the algorithm progresses, the existing harmony vectors may be quite close to the global optima. If we still update the harmony memory aggressively, we may diverge from the optima. As a result, we may want \( HMCR \) to become high and \( PAR \) to be low.

Figure 4.3 illustrates the adaptation pattern of \( HMCR \) parameter on a dataset. In Figure 4.3, the x-axis represents the value of \( HMCR \), and the y-axis shows the percentage of newly generated harmony vectors having better objective function scores than the worst harmony vector in the current harmony memory. We show results for four phases of the execution of the algorithm, denoted as phase 1 (beginning) through phase 4 (end). During this experiment, the \( PAR \) is not changed. From Figure 4.3 we observe that, during the phase 1 the highest fraction of newly generated harmony vectors are better when \( HMCR = 0.15 \). During the phase 2, the peak occurs when \( HMCR \) is around 0.55, at phase 3, \( HMCR \) around 0.7 gives the best results, and at phase 4 (near the end of the harmony search algorithm), the best \( HMCR \) value for generating better harmony vector is about 0.85. The \( PAR \) parameter has a similar trend as the \( HMCR \) parameter.

While the above experiment demonstrates the benefit of increasing \( HMCR \) during
the execution of the algorithm, the desired adaptation pattern of the two parameters
could vary depending upon the dataset and the query. Thus, in our approach, the
adaptation pattern at each iteration is inferred from the statistics of the harmony
vectors we obtained from the previous iteration of the algorithm. We now describe
this method.

4.4.1 Bayesian adaptation Overview

The Bayesian approach is a method which could let us estimate some unknown pa-
rameters (posterior distribution) based on prior knowledge or beliefs (prior distribu-
tion) [14]. In the Bayesian approach, based on a list of observed data $y$, we want to
estimate an unknown parameter $\theta$. The unknown parameter is usually in the form of
a probability distribution $f(y|\theta)$. We could suppose $\theta$ has a prior distribution $\pi(\theta|\eta)$,
where $\eta$ is a vector of hyperparameters. Inference concerning $\theta$ is then based on its
posterior distribution, given by

$$p(\theta|y, \eta) = \frac{f(y|\theta)\pi(\theta|\eta)}{\int f(y|u)\pi(u|\eta)du}$$

In our specific scenario, at each iteration of the harmony search algorithm, the data
we can observe is the harmony parameter values ($HMCR$ and $PAR$) that generate
better stratification candidates. The variable we want to estimate is the adaptation
pattern of these parameters.

Our Bayesian adaptive algorithm works as follows. We assume the adaptation pat-
terns of the two harmony search parameters $HMCR$ and $PAR$ follow the probability
distribution $f_{HMCR}(HMCR|\theta)$ and $f_{PAR}(PAR|\theta)$. The parameter $\theta$ determines the
adaptation pattern of the harmony parameters. $\theta$ is unknown and needs to be esti-
mated. The probability distribution is detailed in Section 4.4.2. Next, we represent
our belief in $\theta$ as a prior probability distribution. This is detailed in Section 4.4.3.
In our method, we take a pilot sample from all possible values of the harmony parameters $HMCR$ and $PAR$, and select the ones, denoted as $HMCR_{opt}$ and $PAR_{opt}$, which generate the best stratification candidate with respect to the current harmony memory. $HMCR_{opt}$ and $PAR_{opt}$ is our observed data. Based on our observed data $HMCR_{opt}$ and $PAR_{opt}$, we update the value of the unknown parameter $\theta$, i.e., compute the posterior distribution. Finally, suppose the estimated value of the unknown parameter $\theta$ is $\hat{\theta}$, then the new adapted harmony search parameters $HMCR$ and $PAR$ for the next iteration of the algorithm could be computed from the probability distribution $f_{HMCR}(HMCR|\hat{\theta})$ and $f_{PAR}(PAR|\hat{\theta})$.

We next describe our Bayesian adaptive approach for estimating $HMCR$ in detail. The adaptation for $PAR$ parameter can be performed similarly.

### 4.4.2 Determining Probability Function $f_{HMCR}(HMCR|\theta)$

To apply Bayesian approach, we need to find a probability distribution which could reflect the pattern of the change of $HMCR$ during the entire procedure of harmony search. Learning from Figure 4.3, we know that the change of $HMCR$ has the following features. First, $HMCR$ ranges from 0 to 1, and second, $HMCR$ starts with small value and then increase along with the iterative procedure of harmony search. In statistics, the Beta distribution best mimics the above two features. The Beta distribution is a family of continuous probability distributions defined on the interval $(0, 1)$ parameterized by two positive shape parameters, typically denoted by $\alpha$ and $\beta$. Figure 4.4 shows the probability density function of Beta distribution with fixed $\beta = 5$ and varying $\alpha$ from 2 to 30.

Therefore, we use a Beta distribution with fixed $\beta$ value to represent the probability function $f_{HMCR}(\cdot)$. The change of the parameter $\alpha$ of the Beta distribution reflects the change of $HMCR$ during the iteration of the harmony search algorithm.
Thus, the parameter \( \alpha \) of the Beta distribution is the unknown parameter \( \theta \) we need to estimate in our Bayesian adaptive method, and we use \( \theta \) to represent the first parameter of a Beta distribution. Thus, to summarize, the probability function

\[
    f_{HMCR}(HMCR|\theta) = \text{Beta}(\theta, \beta)
\]

4.4.3 Determining Prior Distribution and Bayesian Updates for \( \theta \)

In Bayesian analysis, the value of \( \theta \) is estimated after the values of \( HMCR \) are observed. The prior distribution of \( \theta \) expresses one’s belief about \( \theta \) before the data is observed. In other words, in our problem, the prior distribution indicates our belief on the value of \( \theta \) even before the execution of the harmony search algorithm. We know that, for a Beta distribution with fixed \( \beta \) parameter, the shape of the Beta distribution is determined by the relation between the values of \( \theta \) and \( \beta \). If \( \theta < \beta \), the peak of the Beta distribution occurs before \( HMCR = 0.5 \), and if \( \theta > \beta \), the peak of the Beta distribution occurs after \( HMCR = 0.5 \). Furthermore, from Figure 4.3, we know that, roughly, \( HMCR \) changes from small values (smaller than 0.5) to large values (larger than 0.5) during the execution of the harmony search algorithm. This suggests that a reasonable prior belief for \( \theta \) will generate small \( HMCR \) values. Formally, the prior distribution of \( \theta \) is then:
\[ \pi(\theta) = \text{Uniform}(0, \beta) \]

Now, for the parameter \(HMCR\), we have the probability function \(f_{HMCR}(HMCR|\theta)\) and the prior distribution \(\pi(\theta)\). The posterior estimation for the unknown parameter \(\theta\) could be computed using the following expression [14]

\[
E^{\pi(\theta|HMCR)}[\theta] = \frac{\int \theta f_{HMCR}(HMCR|\theta)\pi(\theta)d\theta}{\int f_{HMCR}(HMCR|\theta)\pi(\theta)d\theta}
\]

4.5 Evaluation

In this section, we evaluate our Bayesian adaptive harmony search stratification algorithm. First, we examine the performance of our algorithm using datasets with different correlation between the auxiliary and the selective attributes, as well as using queries with different selectivity. Second, we compare the Bayesian adaptive version of our algorithm with a non-adaptive version of the algorithm. Finally, we compare our method with three other existing sampling methods. Here, we focus on two aspects: total sampling cost for obtaining the same number of samples meeting the selection predicate, and the accuracy achieved with the same sampling cost.

4.5.1 Data Sets

We have used 7 datasets, including five synthetic datasets and two real data sets.

**Synthetic Data sets:** These were generated using MINITAB\(^2\), a statistical software package. We generated five datasets. As mentioned in Section 4.2, our method performs stratification on the selective attribute based on an auxiliary attribute, and therefore, the correlation between the auxiliary attribute and the selective attribute is

\(^2\text{http://www.Minitab.com}\)

102
important for our algorithm. These five synthetic datasets have different correlation values, i.e., 1, 0.85, 0.7, 0.5, and 0.3, respectively. Each synthetic dataset has 100,000 data records and 3 attributes, which are used as the auxiliary attribute, the selective attribute, and the target attribute, respectively.

**US Census Dataset:** The Census dataset, denoted as IBQ, comprises of the 2002 US Economic Census data on Wholesale Trade Product Lines listed by the Kind of Business. This dataset can be downloaded at American FactFinder². We have 6 attributes and 24984 data records. We use profit as the target attribute, sale as the selective attribute, and number of establishments as the auxiliary attribute. An example low selectivity query that we could issue on IBQ dataset is “find the average profit of all kind of business whose sale amount during the year is smaller than $10,000.” Suppose sale is not directly queriable, we need to answer the query based on the stratification generated for the number of establishments attribute. The correlation between the auxiliary attribute (number of establishments) and the selective attribute (sale) is 0.7.

**Yahoo! Auto Data set:** The Yahoo! Auto dataset, denoted as Auto, comprises of the data crawled from http://autos.yahoo.com/. Particularly, it includes the data on used cars from any model located within 50 miles of a zipcode address. This yields a dataset with 4000 data records. We consider price as the target attribute, mileage as the selective attribute, and year as the auxiliary attribute. An example low selectivity query that we could issue on Auto is “find the average price of the used cars with mileage smaller than 5000 miles.” The correlation between the auxiliary attribute (year) and the selective attribute (mileage) is 0.56.

²http://factfinder.census.gov/
4.5.2 Evaluation of Our Approach HarmonyAdp

In this experiment, we evaluate the performance of our approach on datasets with different correlation and queries with different selectivity. We first briefly describe how our experiment is conducted. For each query, we first find the stratification of the dataset using our method. Then, we draw a sequence of 5 independent random stratified samples using the sample allocation method proposed in Section 4.3.3. The query result is estimated independently using each stratified sample set. Finally, the average of the 5 independent estimation results are reports as the final estimation. All queries considered in our experiments are low selectivity queries with an aggregation function that averages the target attribute.

Throughout our experiment, we use Absolute Error Rate (AER) as our evaluation metric. AER reflects the accuracy of an estimator. For a variable with true value $\theta$ and estimated value $\hat{\theta}$, the AER of the estimator $\hat{\theta}$ is $AER(\hat{\theta}) = \frac{|\hat{\theta} - \theta|}{\theta}$.

Figure 4.5 shows the result on the 5 synthetic datasets, and Figure 4.6 shows the result on the 2 real world datasets. In Figure 4.5, each sub-figure shows the result on a dataset with different data correlation ranging from 1 to 0.3. The x-axis represents the number of iterations performed in the HarmonyAdp algorithm, and the y-axis is the AER of the query result estimation obtained. In this experiment, we consider three query selectivity values, which are 1%, 0.1%, and 0.01%.

From Figure 4.5, we have the following observations. First, as expected, with the increase in the number of iterations, the accuracy of the estimation results becomes higher for all correlation cases and all query selectivity values. When the number of iterations is around 40, corresponding to about 2000 samples (about 2% of dataset size), the estimation error rate for 1% selectivity query is below 5% for all cases, and even for 0.01% selectivity queries, the error rate is below 15%. This shows that our method HarmonyAdp is very appropriate for answering low selectivity queries.
Figure 4.5: Evaluating HarmonyAdp on Different Datasets and Queries on Synthetic Data

Figure 4.6: Evaluating HarmonyAdp on Different Datasets and Queries on Real World Data
using limited number of samples (i.e. only 2% of total size of the data). Second, for all data correlation cases, while the estimation accuracy decreases between the 1% and 0.01% selectivity queries, the difference between the error rate of 1% and 0.01% selectivity query is not significant. This shows that our approach is robust with respect to different query selectivity levels. Third, we observe that the error rates for queries of all selectivity levels do not vary too much across datasets with different data correlation. Even for a very low correlation, for example 0.3, the error rates achieved when the number of iteration is 40 range from 5% to 18% for different query selectivity. This shows that our method is also robust in terms of data correlation.

In Figure 4.6, the left sub-figure shows the result on Auto dataset, and the right sub-figure shows the result on IBQ dataset. Here, we consider three query selectivity levels 2%, 1%, and 0.5%. Here, we have similar observations as in Figure 4.5. For the Auto dataset, when the number of iteration is about 20 (7% of total data size), the error rates are always below 13% for all selectivity levels, and the error rates are about 8% for the 2% selectivity query. For the IBQ dataset, when the number of iteration is about 40 (8% of total data size), the error rates are always below 16% for all selectivity levels, and the error rates are about 10% for 2% selectivity query. The correlation of the IBQ dataset is about 0.5, which is not very high. This further shows that our approach works quite well on real a dataset with relatively modest correlation.

4.5.3 Comparision Between Adaptive and Non-Adaptive Versions

In this experiment, we compare the performance of our Bayesian adaptive version of the algorithm, HarmonyAdp, with a non-adaptive version of the algorithm. In the non-adaptive version of the algorithm, denoted as SmartFixed, the $MHCR$ and
$PAR$ parameters are fixed to 0.8 and 0.5 respectively. These two parameter values are suggested (and used) in the original harmony search algorithm publication [91].

Figure 4.7 shows the results on all combinations of correlations and query selectivity values. From all the cases shown in Figure 4.7, we observe that, when the number of iteration is small, the error rate of the adaptive version is lower than the non-adaptive version, especially for queries with extreme low selectivity (such as 0.01%). However, with the increase of the number of iterations, the performance of the non-adaptive version becomes close to the adaptive version. This is an expected result, since the goal of the adaptive version is to achieve a good stratification faster. We could observe exactly the same pattern in Figure 4.8. As introduced in Section 4.4.1, we need extra samples to perform parameter adaptation, as a result, in this experiment, we also compare the total sampling cost of HarmonyAdp with SmartFixed. We observe that the estimation accuracy of HarmonyAdp achieved at 10 iterations is very close to the estimation accuracy of SmartFixed achieved at 20 iterations. For the synthetic data sets, the sample size difference between 10 and 20 iterations is 500, and the sample size used in parameter adaptation is 340. As a result, HarmonyAdp has a 32% reduction in cost to achieve the same quality. Similarly results are also observed in the two real world data sets.

4.5.4 Comparison with Three Other Methods

We now compare our method with three other methods. They are Leaps and Bounds stratification method [57] (denoted as L&B), Dalenius and Hodges stratification method [36, 37] (denoted as D&H), and No stratification (Random Sampling) method (denoted as Random). The L&B and D&H methods are classical optimal stratification methods from statistics literature and are both based on an auxiliary attribute.

**Leaps and Bounds (L&B):** This method assumes that the auxiliary attribute $x$
Figure 4.7: Compare Adaptive and Non-Adaptive Algorithm on Synthetic Data

Figure 4.8: Compare Adaptive and Non-Adaptive Algorithm on Real World Data
is strongly correlated with \( y \). Besides the correlation assumption, the L&B method is based on two more important assumptions, which are uniform distribution within stratum, and equal coefficients of variance in each stratum of the auxiliary attribute. While these three assumptions lead to a much simpler method, which does not require iteration. According to L&B, a L-stratum stratification on an auxiliary attribute is determined by the following stratum boundaries

\[
k_h = ar^h, (h = 0, 1, \ldots, L)
\]

**Dalenius and Hodges (D&H):** As in the L&B method, the basic assumption of the D&H method is the existence of strong correlation between the auxiliary attribute and the unknown attribute. Furthermore, the D&H method also critically depends on the assumption that the distribution of the auxiliary attribute in each stratum is approximately uniform.

**No Stratification (Random):** This is a very simple method which does not stratify the dataset. Instead, we just select a random sample from the entire dataset and perform the estimation.

We conduct two experiments to compare our method HarmonyAdp with the these three methods. The first experiment is as follows. To answer low selectivity queries
on the deep web, we will obviously prefer a method that can be more effective in obtaining records satisfying the selection predicate (referred to as *useful samples*). This is because the number of useful samples directly impacts the accuracy of the estimation, and because total querying cost over the deep web needs to be minimized. Therefore, in this experiment, we compare the total number of samples that need to be selected (i.e., the sampling cost involved), to obtain the same number of useful samples, across the four methods. For HarmonyAdp, the total number of samples includes the samples used in training our model and query estimation. Our goal is to show that HarmonyAdp is a useful method despite the sampling involved while training our model.

Figure 4.9 shows the results on synthetic data. Here, the number of useful samples obtained is fixed at 25. The Y-axis in Figure 4.9 is in logarithmic scale. We have the following observations. First, for *simpler* queries (10% selectivity), all four methods use nearly the same number of total samples to obtain 25 useful samples. Second, for more *challenging* queries (selectivity smaller than 10%), our algorithm only needs about one-tenth of the total number of samples to achieve the same number of useful samples. Third, with reducing selectivity of the queries, L&B, D&H, and the Random
method have a linear increase in the total number of samples needed. However, the increase in total number of required samples is very small for our method. This shows that our stratification method is very effective for selecting samples satisfying the low selectivity clause. Figure 4.10 shows the results on real world data. For the Auto and IBQ dataset, the number of useful samples is fixed to be 10 in the experiment. We observe a similar pattern, i.e., our method always uses the least number of total samples to obtain a specified number of useful samples. Finally, in this experiment, we also compared the AER values achieved by the four methods when the number of useful samples is the same. As expected, all four methods obtain similar accuracy when aggregation is performed over the same number of useful samples.

In the second experiment, we compare the estimation accuracy obtained from the four methods when the total sample size is fixed to be the same for the four methods. For our method, the total sample size includes the samples used in training the harmony search model and the samples used in estimating query results. However, once the search model has been trained, we obtain more useful samples from fewer total number of samples. Figure 4.11 shows the results on synthetic datasets. Here, the sample size is fixed to be 2000 for all four methods (2% of total data size). Specifically, for our HarmonyAdp method, among the 2000 samples, 1500 are used in training, and 500 are used in query estimation. For other three methods, all 2000 samples are used in query estimation. We have the following observations. First, when the query selectivity is 10%, all of the methods achieve good results, across different correlation levels. Second, with a decrease in query selectivity, L&B, D&H and Random methods degrade severely, but our method only incurs a small reduction in accuracy. For 0.1% selectivity queries, HarmonyAdp outperforms other three method by a factor of 5 on the average. For 0.01% selectivity queries, the error rate of our method is consistently lower than 18%. Figure 4.12 shows the results on
two real world datasets. For the Auto dataset, the sample size is fixed to be 300 (7% of the entire data) for all methods. Among these 300 samples, 50 samples are used for query estimation in HarmonyAdp. For the IBQ dataset, the sample size is fixed to be 2500 (10% of the entire data), and among the 2500 samples, 500 samples are used for query estimation in HarmonyAdp. From Figure 4.12, we can observe a similar pattern as in Figure 4.11.

It may be argued that comparison of our method with L&B and D&H methods is not a very fair one, since the latter two methods involve assumptions which did not hold true on these datasets. Similarly, random sampling is a very simple method, and...
it is not surprising that it does not do well on low selectivity queries. The primary conclusion, however, that we will like to make from our experiments is as follows. HarmonyAdp is the first effective method for answering low selectivity queries on data sources that allow only limited data accesses, and where the queriable attribute is not necessarily highly correlated with the selectivity attribute. Our experiments have demonstrated the robustness of our method when selectivity and/or correlation are decreased.

4.6 Summary

In this chapter, we have studied the problem of estimating the result of an aggregation query with low selectivity on data sources that only support limited data access. This work has been motivated by the emergence of deep web data sources, and the need for supporting a rich set of queries over them.

The key challenge we have addressed is of stratification when the selective attribute is not directly queriable. Thus, we need to perform stratification on a queriable attribute, which may not necessarily be highly correlated with the selectivity attribute. We have developed a guided search method, which is motivated by the recently developed Harmony search model, to stratify the queriable attribute in a way that useful samples for a low selectivity query can be effectively obtained. We have mapped the stratification problem to Harmony search, have developed an objective function for guiding the search for low selectively cases, and further, have introduced a Bayesian dynamic adaptation of the two key parameters in harmony search.

We have evaluated our method using both synthetic and real datasets. We show that the query estimation result using the stratification generated from our approach have a high accuracy (about 95%) for low selectivity queries, while querying only 2% of the data. The accuracy achieved using our method is higher than 85% even
for extremely low selectivity queries (selectivity 0.01%). Furthermore, our method is robust even when the correlation between the auxiliary attribute and the selective attribute is relatively low. We have also compared our approach with three other known methods. As these methods involve assumptions that did not hold true on the datasets we considered, not surprisingly, our method outperformed these methods on low selectivity queries by a factor of 5 on the average. Specifically, we obtained better accuracy with the same sampling cost, and incurred lower costs for obtaining the same number of useful samples. Overall, we have shown that our method is the first effective method for answering low selectivity queries on data sources that allow only limited data accesses, and where the queriable attribute is not necessarily highly correlated with the selectivity attribute.
CHAPTER 5
QUERY DEEP WEB DATA SOURCES: A STRUCTURED KEYWORD QUERY APPROACH

As we have mentioned in Chapter 1, the common way of accessing data in deep web data sources is through standardized input interfaces. These interfaces, on one hand, provide a very simple interface, and do not require that the users know any specific query language and/or the schema(s) of the underlying databases. On the other hand, these interfaces significantly constrain the types of queries that could be automatically executed. For example, in the biological domain, most data sources only have one text box in their input interfaces. Such text boxes only accept simple text, i.e., the users cannot specify any constraints, aggregation functions, or nested queries. Furthermore, many useful queries involve accessing information from multiple data sources, and currently, there is no available support for this class of queries. Thus, it is desirable that our system could support keyword search query over deep web data sources.

5.1 Motivation

A popular trend in data dissemination involves online data sources that are hidden behind query forms, thus forming the deep web. Hundreds of large, complex, and in many cases, related, deep web data sources have become available. The common way
of accessing data in deep web data sources is through simple text interfaces. These interfaces, on one hand, provide a simple way for users to access the data, without the knowledge of a specific query language or the schemas of the underlying databases. On the other hand, these interfaces significantly constrain the types of queries that could be automatically executed. For example, in the biological domain, most data sources only have one text box in their input interfaces. Such text boxes only accept simple text, i.e., the users cannot specify any constraints, aggregation functions, or nested queries. Furthermore, many useful queries involve accessing information from multiple data sources, which makes the problem even harder. However, currently, there is no available support for answering complex queries over deep web data sources.

Recently, two ways to deal with deep web data have been articulated, which are, surfacing and virtual integration [98]. An intuitive way of supporting complex structured queries over deep web data is to surface the data and store it in a relational database. Then, standard query languages and techniques can be used. However, data in many deep web data sources is continually updated. Thus, the surfacing approach would lead to a data consistency problem.

In the virtual integration approach, a mediated schema for the data sources in a domain can be created by schema matching techniques [60]. Then, users can pose queries over the mediated schema that is exposed to them as a web form. The mediator approach has been very well studied, prior to the interest in the deep web. Some of the very well known examples of mediator systems that have been built are SIMS [5], Information Manifold [88], TSIMMIS [51], and MedMaker [104]. A number of recent efforts on building deep web querying systems can also be viewed as mediator systems. Among these, the systems described in [17, 118] mainly deal with optimizing the execution of queries on the web, but do not consider query planning for queries
expressed in a high-level, highly-expressive query language. The systems proposed in [84, 127, 118] have query planning algorithms for answering queries over the deep web, but they cannot support aggregation or nested queries.

5.1.1 Motivating Examples

This paper develops enabling techniques for supporting a high-level, highly expressive query language on the deep web. The following two motivating examples illustrate the need for, and challenges in, supporting advanced queries over the deep web.

Motivating Example 1: Suppose we are interested in Single Nucleotide Polymorphisms (SNPs), which are particularly promising for explaining the genetic contribution to complex diseases [2]. Biologists have identified that the gene $X$ and the protein $Y$ are contributors of a disease. Now, they want to examine the SNPs located in the genes that share the same functions as either $X$ or $Y$. Particularly, for all the SNPs located in each such gene functions similar to either $X$ or $Y$, and those that have a heterozygosity value (a measure of the genetic variation) greater than 0.01, biologists wants to know the maximal allele frequency in the Asian population.

The above query can be answered using two nested queries (highlighted by a dotted rectangle in Figure 5.1) and one main query. From the two nested queries, we could obtain the names of the genes sharing similar functions as either $X$ or

Figure 5.1: SQL Query for the Motivating Example
Y. Then for each such gene, the main query, which is an aggregation query, finds the corresponding maximal SNP frequency data. We express this query in SQL-like fashion in Figure 5.1.

In practice, however, answering this query needs data from multiple different deep web data sources. Furthermore, this query requires an aggregation function, a group by operation, value constraints and even nested queries. Specifically, we need to take the following steps, which are also shown graphically through a query plan in Figure 5.2.

1. The first sub-query: find the genes that have the same function as the gene X (query-plan part 1, sub-figure (a), Figure 5.2).

2. The second sub-query: find the genes that have the same function as the protein Y (query-plan part 2, sub-figure (a), Figure 5.2).

3. The main query: for each gene obtained from the steps 1 and 2, find all the SNPs, filter out the SNPs with Heterozygosity value smaller than 0.01, and find the maximal allele frequency in Asian population (query plan part 3, sub-figure (a), Figure 5.2).

Specifically, NCBI Gene and Human Protein data source takes gene X and protein...
Y as input, respectively. The names of the genes with the same function as X or Y will be obtained from the GO data source in the query plan parts 1 and 2. Taking the genes obtained from GO, and using the SNP500Cancer data source, we can find the human SNPs located on these genes. Finally, using the dbSNP data source, we can find the SNP frequency and heterozygosity data. Because the input interface of dbSNP does not support aggregation operation, our system needs to generate a plan to perform the aggregation computation on the allele Asian frequency of the SNPs obtained, while filtering out all SNPs with heterozygosity value smaller than 0.01.

**Motivating Example 2:** A travel agent in a mid-west city X wants to advertise new travel destinations on the west coast. He is interested in finding the answers to the following query “find all west cost cities that have an average airfare (from city X on particular dates) that is lower than the average airfare to Los Angeles”. To answer this query, we need to perform two aggregations. Furthermore, X to Los Angeles is a sub-query that provides a condition value for the main query.

### 5.1.2 Challenges and Contributions

From the above two examples, we observe that to answer complex queries on the deep web, we need to address the following issues. First, we need to know which data sources are relevant to the query and identify the inter-dependence between data sources. For example, in the query plan in Figure 5.2, the NCBI GENE data source must be queried before the GO data source because the output from NCBI GENE is used as the input of GO. Second, the input interfaces of most deep web data sources only accept simple text queries, when generating query plans, the system must handle the complex structures in the query such as aggregation functions and nested queries. Furthermore, we need to generate the query plans with the least querying cost (Section 5.2.2). Finally, during query plan execution, data obtained from deep
web data sources must be transmitted through a network. As a result, we need query execution optimization techniques to speedup query execution.

In this chapter, we develop and evaluate a framework for querying a set of integrated deep web data sources. Currently, we support three types of advanced query: 

*SPJ queries, aggregation queries, and nested queries.* We use SQL to illustrate our algorithms and examples, though our techniques can also be applied on other query languages with similar features. Our approach for supporting these queries is as follows. 1). If the query contains nested queries, we divide the original query into a list of *simple queries*, i.e., the queries without nested queries. Then, for each *simple query*, we extract the search terms and necessary predicates to understand the query type. 2). We use our novel query planning algorithms to generate a query plan for each simple query. The three dotted rectangles in Figure 2(a) show the three query plans for three simple queries of the query in the first motivating example. 3). The query plans for all simple queries in the original query are *combined* and *merged* together to form the final query plan. For example, the Figure 2(a) shows the *combined* query plan for motivating example 1 and the Figure 2(b) shows the final plan after merging similar data sources *NCBI Gene* and *GO* from the query plans part 1 and 2. 4). Finally, several optimization techniques that we have developed are applied to speedup the execution.

Overall, the main contributions of this paper can be summarized as follows:

1. We have formulated novel query planning problems, for executing different types of complex structured queries over multiple inter-dependent deep web data sources.
2. We have developed query planning algorithms to generate query plans for different types of advanced queries, including SPJ queries, aggregation and nested queries.

Compared with the existing work on mediator systems [5, 51], our work is distinct in two important ways. First, we support more advanced queries, i.e. aggregation
and nested queries. Second, we have a different cost model, i.e., in the deep web, we want to find the query plan with the minimal size, and therefore, our query planning problem formulation and algorithms are very different.

3. Several optimization techniques, including query plan merging and grouping optimization, have been developed to accelerate query execution. 4. In the experiments, we show that for over 90% of the experimental queries, the execution time and result quality of the query plans generated by our algorithms are very close to that of an exhaustive search algorithm. Furthermore, our optimization techniques outperform an existing optimization method in terms of both reduction in transmitted data records and query execution speedups.

5.2 Query Planning Problem: Formulation and Algorithms

In this section, we first give an overview of the query planning problem for the query types we are considering. Then, we formally state the query planning problem for the three types of queries, and present the query planning algorithms for each of these.

5.2.1 Query Planing Problem Overview

Given a query \( Q \), If it contains nested queries, we divide \( Q \) into a list of simple queries, i.e., the queries without nested queries. Then, we generate a query plan for each simple sub-query of \( Q \). Finally, the query plan for \( Q \) is obtained by combining and merging all the query plans thus generated.

In our query planning problem, we consider three types of simple queries. The first type is the ordinary SPJ query, which is a select-project-join (SPJ) query that neither includes aggregation nor a nested sub-query, but could have condition predicates. The second type is an aggregation query, which is a query with an aggregation function and/or group-by operation. The third is nested query.
Each type of simple query has specific requirements for a valid query plan. Thus, they need to be treated separately. In the rest of this section, we will present algorithms for each of these three types of queries. The overall query planning algorithm is based on invoking these algorithms, and a *combine-merge* function which will be presented in Appendix 5.4.

### 5.2.2 Query Planning for Ordinary SPJ Query: Formulation and Basics

Ordinary SPJ query has the standard *select-from-where* format. We consider the following example of an ordinary SPJ query.

Example 3

```sql
SELECT s.Frequency FROM SNP s, GENE g WHERE s.Heterozygosity>0.01 AND s.genename=g.genename AND g.genename='Z'
```

This ordinary SPJ query has the following intention: for all the SNPs located in gene *Z* and with Heterozygosity value greater than 0.01, give the Frequency values of such SNPs. We extract the *search terms* from the *select* and *where* clause in the ordinary SPJ query. *Search terms* can be divided into two sets, the *entity terms* and *attribute terms*. An *entity term* is used to *initiate* the answering of the query. In the deep web scenario, when querying a data source, a user must specify values for the input attributes of the data source to initiate the search, and these values are called the *entity terms*. We define the search term(s) in the *where* clause with a value assignment in a SPJ query as the *entity term(s)*. In the above example, the gene name *Z* is an *entity term*. Considering the deep web scenario, throughout this paper, we refer to the SPJ queries with *entity terms* (value assignments in the *where* clause) as *ordinary SPJ queries*. The case of SPJ queries without *entity terms* is discussed in Appendix 5.6. Besides the *entity terms*, all other search terms in the *select* and *where* clauses are *attribute terms*. This is because in answering the query, we need to obtain the *values* of these *attribute terms*. In the above example, there are two *attribute
terms, which are Frequency and Heterozygosity. The value of Heterozygosity is used as a condition filter and the value of Frequency is the final target of the query.

We could say that an ordinary SPJ query finds the data records that are specified by the entity term(s), and then obtains the values of the attribute terms from these data records. For the attribute term that is a value constraint, such as Heterozygosity > 0.01, we could use it as a data record filter during the plan execution.

Capturing Data Source Dependencies
Before formally stating the query planning problem, we first define the data source dependency graph. We consider deep web data sources are connected by the inter-dependence between them and form a dependency graph. Graph nodes represent data sources and edges represent the inter-dependency relation between pairs of data sources. If there is a dependency edge pointing from data source node u to data source node v, it shows that the output from data source u would be used as the input values for data source v. we call u is the parent of v, and v is the dependent of u. In Figure 5.3, we show the inter-dependency relation between 5 data sources.

Some data source dependencies are multi-source, i.e. the input of a data source depends on the output from multiple data sources, as a result, some data source node has composite parents. For example, in Figure 5.3, two highlighted edges linked by an arc shows that dbSNP and Entrez protein form a composite-parent for BLAST, which means that to be able to query BLAST, one needs to query both dbSNP and Entrez Protein first.

Problem Formulation and Cost Model
We do not consider the possibility of having to filter data tuples, as it is handled during plan execution. Thus, given an ordinary SPJ query Q and a data source dependency graph DG introduced as above, the query planning problem is formulated as follows.
We want to find a subgraph $SubG$ from $DG$, such that the set of all output attributes from the nodes in $SubG$ covers all attribute terms in $Q$, and the set of all input attributes of the nodes without incoming edges in $SubG$ covers all entity terms in $Q$. The data source nodes covering attribute terms are called target nodes, and the data source nodes covering entity terms are called starting nodes.

The reason why such a subgraph $SubG$ forms a valid query plan for $Q$ is as follows. For an attribute term $t_a$ in $Q$, we want to obtain its value from certain data sources, so we need at least one node in $SubG$ that outputs (or covers) $t_a$. For an entity term $t_e$, $t_e$ helps initiate answering of $Q$. Thus, we need the input of the starting nodes in $SubG$ to cover $t_e$. Let us consider the query in Example 3. Because the entity term is the gene name $Z$, and the two attribute terms are Heterozygosity and Frequency, we want to find a subgraph which could connect a data source node, which accepts Gene Name as input, with other data sources, which output Heterozygosity and Frequency, as the query plan. The join operation in an SPJ query is realized by the inter-dependency edges between data sources in the query plan.

Clearly, there may be multiple subgraphs of the dependency graph that satisfy the conditions for a valid query plan we have stated above. Among these query plans,
we want to identify the query plan with the \textit{least execution} time, and the one which is likely to give the \textit{highest quality} of results. In our approach, we combine these two considerations into a single \textit{cost model}. Though the details of these cost models are available from an associated technical report \cite{126}, the basic ideas are as follows. For every data source \( D \) included in a candidate query plan, we associate an \textit{access cost}, and a \textit{quality cost}. The access cost includes the response time of \( D \), which is obtained by \textit{off-line data source profiling}, i.e., by issuing pilot queries on a data source, and obtaining the average response time. In the future, our work can incorporate more advanced access cost estimation methods \cite{56, 17}. The second component, \textit{quality cost}, captures the fact that different data sources in a domain may contain similar data, but with different data quality or a higher trust factor \cite{15}. For example, a data source may be likely to return more data elements in response to a given query, or alternatively, data sources maintained by large and reputed institutions are likely to be trusted more by the users. To capture this together with the access cost, we associate a higher cost with data sources that may have a lower quality or may otherwise not be preferred by the users. This information is clearly domain specific and our system assumes that this information is captured and is available.

Now, our query planning problem becomes: \textit{We want to find a subgraph} \( \text{SubG} \) \textit{from} \( \text{DG} \), \textit{such that the set of all output attributes from the nodes in} \( \text{SubG} \) \textit{covers all attribute terms in} \( Q \), \textit{and the set of all input attributes of the nodes without incoming edges in} \( \text{SubG} \) \textit{covers all entity terms in} \( Q \), \textit{and furthermore, among all subgraphs meeting these requirements,} \( \text{SubG} \) \textit{has the lowest sum of access and quality costs}.

Given the query planning problem formulated as above, we have established the following result.

\textbf{Lemma 5.2.1.} \textit{The query planning problem for an ordinary SPJ query is NP-hard.}
5.2.3 Bidirectional Query Planning Algorithm

To find the minimal cost query plan for an ordinary SPJ query, we have developed a heuristic bidirectional graph search algorithm.

**Algorithm Overview:** A query plan for SPJ query ultimately connects a subset of target nodes with a subset of starting nodes, such that all search terms are covered. We explore the query plan in a bidirectional manner. We perform backward exploration from the target nodes to connect them with starting nodes. To accelerate this process, we also do forward exploration from the starting nodes. To find the query plan with cost as least as possible, we apply the following two heuristics.

**Heuristic 1:** Always try to add the data source with the least cost into the query plan.

**Heuristic 2:** If multiple data sources need to be connected to form a query plan, connecting them via the shortest path.

**Bidirectional Exploration:** Initially we add all starting nodes to a forward exploration queue, and all target nodes to a backward exploration queue. Then, the algorithm tries to find a sub-graph (with minimal cost) to connect the target node set with the starting node set. At each iteration of the sub-graph exploration, the algorithm always selects the node with the least cost, $CN$, from the two queues, using the cost model described above. If $CN$ belongs to the forward queue, all out-going neighbors of $CN$ are explored. If $CN$ belongs to the backward queue, all in-coming parents of $CN$ are explored.

**Edge Exploration:** To build the sub-graph as the final query plan, paths (sequence of graph edges) must be explored to connect target nodes with starting nodes. Here, we apply the second heuristic which is that we always connect pair of nodes through the shortest path. This is realized by modifying the Dijkstra’s shortest path algorithm. Particularly, our modifications are made for handling with composite parent
nodes. This is because the shortest distance between a pair of nodes $u$ and $v$ via a composite node actually depends on the longest path among all the paths connecting $u$ with $v$ via each individual node in the composite node. The edge exploration procedure introduced here is also used in the other two planning algorithms that we describe later in the Section.

**Algorithm Termination:** When every search term can be reached from at least one starting node with finite distance, a query plan has been found.

This algorithm may seem similar to the Steiner tree algorithm used in the context of keyword search on relational databases (Kacholia et al. [81]). Given a set of nodes $N$, the Steiner tree algorithm finds a subgraph which achieves pair-wised connection for nodes in set $N$. However, the connection requirement between the target nodes and the starting nodes we have is different from the steiner tree problem.

### 5.2.4 Query Planning for Aggregation Query: Formulation and Algorithm

Similar to the ordinary SPJ query defined in Section 5.2.2, we extract the attribute terms and entity terms from an aggregation query. For an aggregation query, there can be two special types of attribute terms, the aggregation attribute and the grouping attribute. An aggregation query forms groups based on the grouping attribute and perform the aggregation function on the aggregation attribute for each group. For simplifying our presentation, we assume that we have only one grouping attribute for an aggregation query. We further define a data source whose output covers any aggregation attribute to be an aggregation data source. Similarly, a data source whose output covers the grouping attribute is considered the grouping data source.
Formulation for Aggregation Query

In order to perform aggregation functions on the groups formed by grouping attribute, we must know the mapping between the grouping attribute and the aggregation attributes. Let us consider the following example: we have a list of genes located in chromosome 10, for all the SNPs located in these genes, we want to group the SNPs according to gene name and for each group find the maximal SNP heterozygosity data. We know Gene Name is the grouping attribute and Heterozygosity is the aggregation attribute. In order to perform the grouping and aggregation, we need to know which SNP heterozygosity value maps to which gene. In other words, if SNP heterozygosity and gene data is covered by two data sources, these two sources must be connected, so that we can find the mapping between SNP heterozygosity and genes.

A correct query plan for this query is shown in Figure 5.4(a). Using chromosome 10 as the input, from data source A, we could obtain a list of genes located in the chromosome 10. Then, from the data source B, we obtain the SNPs located in each such gene, and further from data source C, we obtain the heterozygosity value for each such SNP. In this query plan, the data source A is the grouping data source since it covers Gene Name, which is the grouping attribute. The data source C is the aggregation data source, because it covers Heterozygosity, which is the aggregation attribute. We observe that the path connecting the grouping data source A and the aggregation data source C clearly shows the mapping between heterozygosity values and genes, i.e., which heterozygosity value comes from which gene. Figure 5.4(b) shows another query plan, in which the aggregation data source (F) is not connected with the grouping data source (E). If this query plan is to be used, we can obtain gene name and SNP heterozygosity data located in the chromosome 10 from E and F, respectively, but we cannot know which SNP heterozygosity data comes from which.
gene. As a result, we cannot perform the grouping operation on Gene Name using the plan in Figure 5.4(b).

**Aggregation Node Connection Property:** We can state that for an aggregation query, a valid query plan must meet the *Aggregation Node Connection Property*, which requires that the *aggregation data source* must be directly or indirectly connected with the *grouping data source* in the query plan.

**Formulation for Aggregation Query:** Given an aggregation query $Q$, the query plan of $Q$ is the minimal cost sub-graph, $SubG$, as formulated in Section 5.2.2, which also satisfies the *aggregation node connection property*. We have shown the following result.

**Lemma 5.2.2.** *The query planning problem for an aggregation query is NP-hard.*

**Center-spread Query Planning Algorithm**

Neither our bidirectional planning algorithm proposed for ordinary SPJ query nor any of the existing Steiner tree algorithm considers the *aggregation node connection property*. We have conducted a pilot experiment to generate query plans for aggregation
queries using the steiner tree algorithm. For 16 aggregation queries we experimented, the steiner tree algorithm only yields correct query plans for 5 of them (lower than 30%), and furthermore, these query plans have high cost. This shows the necessity for us to develop a new algorithm for generating query plans for aggregation query.

To ensure the aggregation node connection property in a query plan, we start building a query plan from the aggregation data sources and gradually explore other data sources to the query plan if they satisfy the aggregation node connection property. This procedure continues until all search terms are covered by data sources explored and reached from some starting nodes with finite distance. In other words, we consider the aggregation data sources as seeds and explore the graph to generate the query plan starting from the seeds. We call this algorithm the center-spread query planning algorithm. We describe this algorithm below, and show the pseudo-code in Algorithm 5.3.1.

Algorithm Initialization: Same as the bidirectional algorithm, we have starting nodes (nodes covering entity terms) and target nodes (nodes covering attribute terms) initialized in this algorithm. We have a center spread exploration query CSE. This queue contains the nodes to be explored. Initially, the seed nodes (aggregation data sources) are explored, and we add the data sources that are directly connected with at least one seed node to the CSE queue, because these data sources are going to be explored in the following steps.
5.3 Center-Spread Algorithm Pseudo-code

Algorithm 5.3.1: Center-spread\((Q)\)

1 Initialiation and explore seed nodes
2 foreach aggregation data source node seed
3 \(CSE.add(\text{Neighbors}(\text{seed}))\)
4 while \(CSE \neq \Phi\)
5 select the node with lowest cost from \(CSE\)
6 denot it as \(\text{ProbeNode}\)
7 if \(\text{ProbeNode}\) satisfies Aggregation Node Connection Property
8 add \(\text{ProbeNode}\) to query plan
9 add \(\text{Neighbors}(\text{ProbeNode})\) to \(CSE\)
10 foreach pair of nodes in current plan
11 update shortest path information
12 foreach node \(u \in\) query plan
13 if no shortest path going through node \(u\)
14 delete \(u\) from query plan
15 delete \(\text{Neighbors}(u)\) from \(CSE\)
16 if find path with finite distance to every search term
17 from a subset of starting nodes
18 we find query plan for the query
19 if No nodes in \(CSE\) satisfies Aggregation Node Connection Property
20 Abort and inform the user

Plan Exploration: In each iteration of the algorithm, using heuristic 1, we choose the lowest cost data source, \(\text{ProbeNode}\), from the center spread exploration queue. We check whether adding \(\text{ProbeNode}\) to the existing query plan respects the aggregation node connection property. If the property is violated, we skip the current \(\text{ProbeNode}\),
and select the second lowest cost node from the $CSE$ queue to repeat the above procedure. Suppose $ProbeNode$ respects the aggregation node connection property, then $ProbeNode$ is explored and added to the query plan. Then we need to update the queue $CSE$. We add the direct neighbors of the node $ProbeNode$ to $CSE$ showing that these nodes can be explored in the future. Because a new node $ProbeNode$ is explored, similar as in the bidirectional planning algorithm, we need to update the shortest path between any other pair of nodes in the current query plan using heuristic 2. After the path updating phase, if a data source $D$ no longer possesses any shortest path to any other node, it should be removed from the plan. If $D$ is removed, $D$’s direct neighbors that are in the $CSE$ queue should also be removed.

**Algorithm Termination:** When all query search terms can be reached from a subset of the starting nodes with a finite distance, a query plan is obtained. During the plan exploration phase, if no data source in the $CSE$ queue can be explored without violating the aggregation node connection property, the algorithm will conclude that the query cannot be answered completely. If this happens, our system would notify the users, since we cannot generate a correct plan for this query.

### 5.3.1 Query Planning for Nested Subquery: Formulation and Algorithm

Figure 5.6 shows one of the nested sub-queries from the first motivating example. Here, we want to find the names of the genes that have the same functions as the gene $X$. In this example query, we define the attribute `genename` in the `select` clause of the nested query as the nested attribute. This is because its values will be passed to the outer main query during plan evaluation. We define the attribute `function` in the nested query as the linking attribute. This is because the linking relation between the unknown gene $g$ and the gene $X$ is that they share the same `function`.

We can notice that the nested query in Figure 5.6 is a un-correlated nested query.
This is because we don’t need to take every gene in the main query and evaluate it in the nested query. For this paper, we develop algorithm to handle an un-correlated nested query. The challenges associated with handling a correlated nested query will be discussed in Section 5.6.

**Formulation for Un-Correlated Nested Query**

**Node Linking Property:** Given a un-correlated nested query \( Q \), we extract attribute terms and the entity terms. Note that, the set of attribute terms must contain a linking attribute and a nested attribute. In the query plan of \( Q \), we require the data source covering the linking attribute (denoted as linking data source) to be topologically before the data source covering the nested attribute. This is denoted as the node linking property, which is illustrated using the example below.

Let us consider the query shown in Figure 5.6. A valid query plan for this query is shown in Figure 5.5(a). Using this plan, from the data sources \( A \) and \( B \), we can obtain the functions of the gene \( X \). Next, from the data source \( C \), we can obtain the genes that have the same functions. The plan in Figure 5.5(a) respects the connection through the linking attribute “function” specified by the query. Figure 5.5(b) shows a plan with an error, because the linking data source \( E \) is not connected with the data source \( F \) covering the nested attribute (gene). As a result, although we can obtain the functions of gene \( X \) and genes related with gene \( X \), but there is no way for us to find the relation between the functions and genes. Sub-figure (c) shows another plan with an error, as the linking data source \( J \) is topologically after the data source \( H \) covering the nested attribute (gene). From this plan, we can only know the function of a gene and how this gene is related with the gene \( X \), but “function” attribute here doesn’t play the role of linking, as required in the query.

**Formulation for Un-Correlated Nested Query Planning:** Given a un-correlated
nested query $Q$, we want to find a minimal cost sub-graph, $SubG$, as formulated in Section 5.2.2, which also satisfies the node linking property. We show the following result:

**Lemma 5.3.1.** The query planning problem for a un-correlated nested query is NP-hard.

**Query Planning for Un-Correlated Nested Query**

To ensure the Node Linking Property, we use a variation of the center-spread planning algorithm. In the linkage planning algorithm, we consider the linking data source as the seed data source. We incrementally add data sources to the query plan only when the node linking property is respected.
5.4 Query Plan Combination and Merging

After we generate query plan for each simple query using the algorithms proposed in Section 5.2, we first connect the query plan of the nested simple query with the query plan of the outer query. Then, we need to perform query plan merging.

Merging components of two query plans can help achieve better performance. As shown in the example in Figure 5.2, there can be data sources in the combined plan that could be merged due to similarity among the sub-queries. The main purpose of plan merging is to reduce the transmission cost of a query plan, where the transmission cost is defined to be the total number of data records transmitted during a query plan’s execution [84].

While plan combination can create a correct query plan, merging components of two query plans can help achieve better performance. As shown in the example in Figure 5.2, there can be data sources in the combined plan that could be merged due to similarity among the sub-queries. The main purpose of plan merging is to reduce the transmission cost of a query plan, where the transmission cost is defined to be the total number of data records transmitted during a query plan’s execution [84].

Our planing merging approach is based on a modification of an existing planing merging approach developed by Kementsietsidis et al [84]. In this approach, a plan edge is denoted as \( e = \{n_1, n_2\} \), where \( n_1 \) and \( n_2 \) are the two data sources on the edge. Two edges \( e_1 = \{n_1, n_2\} \) and \( e_2 = \{n_3, n_4\} \) can be merged if the following two conditions hold: 1) the two edges are compatible, i.e., \( n_1 = n_3 \) and \( n_2 = n_4 \), and 2) the ordering of the two edges are respected, i.e., if edge \( e_1 \) and \( e_2 \) are merged, edge \( e_3 \) and \( e_4 \) are merged, and \( e_1 \) topologically proceeds \( e_3 \), then \( e_4 \) should not topologically proceed \( e_2 \).

Given multiple query plans with a list of mergeable edges, Kementsietsidis et al used a *partial order alignment* method to find the optimal merging (the merging
which reduces the most transmission cost). In the partial order alignment method, a *compatibility graph* \( CG = (V, E) \) is built. In \( CG \), each node \( n_{CG} \) represents a pair of mergeable plan edges \( e_1 \) and \( e_2 \). There is an edge \( e_{CG} \) between two nodes \( n_{CG1} = (e_1, e_2) \) and \( n_{CG2} = (e_3, e_4) \) if the two pairs of mergeable plan edges do not map the same data source node to different data sources in the query plan. Then, the optimal merging can be obtained by finding a maximal clique in the compatibility graph \( CG \). The details of this approach can be found in [84].

To further achieve better performance on the deep web, we made two important modifications to the existing approach: *merging condition modification* and *merging algorithm modification*, which are described below.

**Merging Condition Modification:** In the existing method, two plan edges \( e_1 = \{n_1, n_2\} \) and \( e_2 = \{n_3, n_4\} \) are compatible if each pair of ending data sources must be *the same*. This merging condition is too *rigid*, and some mergeable plan edges might be missed as we will shown in our experiment. Consider the two edges \( e_1 = (\text{SNP500Cancer,SeattleSNP}) \) and \( e_2 = (\text{JSNP,SeattleSNP}) \) in Figure 5.7. If we require mergeable edges to have *exactly the same* ending data sources, edges \( e_1 \) and \( e_2 \) cannot be merged. However, in this example, although \( \text{SNP500Cancer} \) and \( \text{JSNP} \) are different data sources, they share data redundancy, i.e., both of them take *gene name* as input and have *SNPID* as output (although *Chromosome* is also an output from \( \text{JSNP} \), it is irrelevant to edge \( e_2 \)). As a result, the *usage* of both pairs of data sources is the same for edge \( e_1 \) and \( e_2 \). Therefore, \( e_1 \) and \( e_2 \) can be merged. Formally, we define two edges \( e_1 \) and \( e_2 \) to be compatible if the *usage*, i.e., the *used* input and output attributes, of the data sources on the edges are the same.

**Merging Algorithm Modification:** In the existing method, since only *exactly the same* data sources can be merged, the *compatibility graph* is unweighted, i.e., we give no preference to different pairs of mergeable edges. But in our case, different data
sources that possibly provide data with different quality, can also be merged. Thus, we need to assign preferences to different mergeable edges.

**Mergeable Edges Weight:** The merge weight of two mergeable edges \( e_1 = (n_1, n_2) \) and \( e_2 = (n_3, n_4) \) is determined by the similarity of each pair of merging data sources, formally, \( weight(e_1, e_2) = Sim(n_1, n_2) + Sim(n_3, n_4) \), where \( Sim(n_i, n_j) \) is determined by an domain ontology.

**Finding Maximal Node-weighted Clique:** With the above mergeable edge weights, the compatibility graph \( CG \) in our scenario is a node weighted graph. Thus, we need to find a maximal node-weighted clique from the compatibility graph to solve the optimal merging. For this purpose, we adopt the *Reactive Local Search (RLS) Algorithm* [13], which finds maximal cliques by stochastically adding and deleting nodes with the largest edge degree to and from the current clique. We make the following modification to the original algorithm: when adding a node to the current clique, among the largest degree nodes, we prefer the node with the *largest node weight*. Similarly, when deleting a node, among the largest degree nodes, we prefer the node with the *smallest node weight*.

---

Figure 5.7: Example for Modified Merging Condition
5.5 Optimization on Query Execution

In this section, we introduce two optimization techniques for query plan execution, which are pipelined aggregation and moving partial grouping-by forward. These two optimization techniques are mainly used for query plans of aggregation queries.

5.5.1 Pipelined Aggregation

We define a query plan for an aggregation query to be grouping-first if the grouping data source topologically proceeds the aggregation data source(s). The query plan in Figure 5.8(a) is a grouping-first query plan. Here, we want to form groups using gene name (A is the grouping data source) and then for each group, perform aggregation function on SNP frequency (dbSNP is the aggregation data source). Suppose we have a gene X which contains 20 SNPs. This optimization will imply that we input the 20 SNPs to dbSNP in a pipelined manner and perform the aggregation incrementally. Pipelined aggregation can reduce the query plan transmission cost by early pruning if the aggregation is involved in a comparison predicate. For example, suppose we want to find the gene group that has a maximal SNP Frequency smaller than 0.6. If we do not use pipelined aggregation, for gene X, we need to issue 20 queries on dbSNP to compute the aggregation function, even if the second SNP in X has a frequency value large than 0.6. Using pipelined aggregation, we issue only 2 SNP queries, as gene X can be pruned.

5.5.2 Move Partial Group-by Forward

We define an aggregation query plan to be aggregation-first if the aggregation data source topologically proceeds the grouping data source. The query plan in Figure 5.8(b) is an aggregation-first query plan, in which we want to form groups using
Suppose in this example, we have 20 SNPs from 2 chromosomes. Without optimization, we issue 20 SNP queries on dbSNP and we obtain the frequency data for these 20 SNPs and 20 gene names, next, we issue 20 gene queries on NCBI Gene to find the 2 chromosomes, and finally, we perform group-by and aggregation. There will be a total of 40 queries issued. But, since dbSNP also provides the gene in which an SNP is located, we could execute this plan in an alternative way. After issuing 20 queries on dbSNP to find the frequency and gene data, we first perform a partial group-by for SNPs on genes. Suppose, the 20 SNPs are grouped into 4 gene groups, and further, we know that the SNPs from the same gene group must map to the same chromosome. We only need to issue 4 gene queries, instead of 20, on NCBI Gene to obtain the chromosome data. The total plan transmission cost is reduced from 40 to 24. We refer to this technique as move partial group-by forward.

The idea of moving partial grouping-by forward was originally proposed in relational database query optimization [32]. Specific to the deep web scenario, we have an aggregation query, and its query plan is aggregation-first. Suppose the grouping data source is GD, the grouping attribute is ga, the aggregation data source is
AD, and the input attribute of the aggregation data source is $aa$. We could perform move partial grouping-by forward method if both the following conditions hold: 1) the aggregation data source $AD$ covers a term $pga$, and there is a 1 to 1 relationship between $pga$ and $ga$ in the user domain, 2) there is a N to 1 relationship between $aa$ and $pga$ in the user domain.

Under this circumstance, we call the term $pga$ a partial grouping-by attribute, and we could do a partial grouping-by operation using $pga$ on data source $AD$ for the aggregation attribute $aa$. In the example above, $pga$ is Gene and the input of $AD$ (dbSNP) ($aa$) is SNP. Since a gene contains multiple SNPs, i.e., N to 1 relation between SNP ($aa$) and gene $pga$, and one gene must be located in one chromosome, i.e., 1 to 1 relation between gene ($pga$) and chromosome ($ga$). As a result, performing partial group-by could reduce transmission cost as shown above.

In order to use the moving partial grouping-by forward technique, we need to understand the relationships between domain terms. This is determined using an domain ontology.

5.6 Extensions and Limitations

This paper has focused on three types of popular database queries. In this section, we briefly describe how our framework can be extended to handle some of other popular query language constructs. We also discuss what kind of database queries cannot be executed on the deep web using our current framework.

5.6.1 Supporting Other Query Operators

The following SQL constructs can be handled by simple extensions to our framework. **Union and OR:** A query with these two operators can be divided into independent sub-queries, each of which can be handled by our current algorithms.
**Order by:** The *order by* construct does not impact query planning. Therefore, supporting *order by* is straightforward.

**Having:** The *having* clause is only used in aggregation queries. We denote the attribute in a *having* clause as the *group filtering attribute*, because it is used to filter the groups formed by the grouping attribute. To perform group filtering, we need to know the mapping between the *grouping attribute* and the *group filtering attribute*, i.e., which value of a *group filtering attribute* comes from which group. Therefore, we require the data source covering the *grouping filtering attribute*, denoted as the *group filtering data source*, must be connected with the *grouping data source*. We call this the *grouping node connection property*. A minor modification to our center-spread planning algorithm can incorporate *having* into our current framework. When we want to explore a *group filtering data source*, we check whether the data source satisfies the *grouping node connection property*.

### 5.6.2 Limitations

Because of the nature of the deep web, we cannot handle the queries that require *data enumeration*, i.e., require that we scan all data tuples stored in a deep web data source. One example is SPJ queries without the entity term (no value assignment in the *where* clause) as mentioned in Section 5.2.2. Consider a simple SPJ query: 

```sql
SELECT * FROM A B WHERE A.a=B.a
```

This is no entity term here (*A.a=B.a* is a join assignment, not a value assignment). As a result, to answer this query, we need to *scan* all data tuples from the data sources *A* and *B*. Similar problem arises from the correlated nested queries described in Section 5.3.1. Consider a correlated query with two tables *A* and *B*, and assume that *A* is the outer query table, and *B* is the inner query table. To answer this correlated query, we need to *scan* every data record in *A*, and compare it against the table *B*. 

141
Data enumeration or scanning can be performed in relational database because the system can access the entire data. However, scanning all data records in a deep web data source requires the enumeration of all possible input value combinations, which is impossible in most cases.

To address this limitation, if we could obtain a representative sample of the data from a data source, we may find an approximate answer to the queries requires data enumeration. In the future, we plan to adapt the existing work on sampling hidden data sources [41] to extend our framework, and approximately answer queries requiring data scans.

5.7 Evaluation

This section describes the experiments we conducted to evaluate our algorithms.

5.7.1 Experiment Setup

Our evaluation was done using 12 biological deep web data sources which include NCBI dbSNP\(^1\), NCBI Gene\(^1\), NCBI Protein\(^1\), NCBI BLAST\(^1\), SNP500\(^2\), Seattle\(^3\), SIFT\(^4\),

\(^1\)http://www.ncbi.nlm.nih.gov/

\(^2\)http://snp500cancer.nci.nih.gov/home_1.cfm

\(^3\)http://pga.gs.washington.edu/

\(^4\)http://blocks.fhcrc.org/sift/SIFT.html
Table 5.2: Examples of Experimental Queries

<table>
<thead>
<tr>
<th>Queries</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>select SNP, gene name, protein sequence, disease</strong> from chromosome c where c.position=10q</td>
<td>SPJ</td>
</tr>
<tr>
<td><strong>select Max(SNP frequency)</strong> from Gene g where g.name=ERCC6</td>
<td>Agg</td>
</tr>
<tr>
<td><strong>select gene ontology</strong> from Gene g where g.name={<strong>select gene name</strong> from Chromosome c where C=10q}</td>
<td>Nested</td>
</tr>
</tbody>
</table>

BIND\(^5\), HGNC\(^7\), ALFRED\(^8\), Human Protein\(^9\), and Uniprot\(^{10}\). The input and output schemas of the data sources were extracted using a previously created wrapper. The dependency graph of the data sources was constructed by analyzing the correspondence between the output and input attributes of different data sources.

Throughout our experiments, 50 queries are used. They are based on our collaboration with a biologist focusing on SNP-related studies [128]. Due to lack of space, we list 3 queries we used in our experiment in Table 5.2, all other queries have a similar structure.

\(^5\)http://www.bind.ca

\(^7\)www.genenames.org

\(^8\)http://alfred.med.yale.edu/alfred/

\(^9\)http://alfred.med.yale.edu/alfred/

\(^{10}\)http://www.uniprot.org/
5.7.2 Query Planning Algorithm Evaluation

In our evaluation, we compare up to three algorithms. The first is one of the three query planning algorithms we have developed, depending upon the type of the query. The second is an exhaustive search algorithm (OPT), which searches the entire sub-graph space exhaustively to find the minimal cost query plan, and has an exponential complexity. Finally, for the SPJ queries, we also compare against the Steiner tree algorithm.

Scalability Evaluation

To compare the scalability between our algorithm and the exhaustive algorithm, we record the query planning time varying the number of data sources. Since our goal was to evaluate scalability, and because we did not need the parsers to obtain the query results, we experimented with upto 75 data sources. The results are shown in Figure 5.9. This figure is plotted with a logarithmic scale, where x-axis is the number of data sources and the y-axis is the query planning time in seconds. We observe that the exhaustive algorithm scales very poorly, and in fact, when the number of data sources is 75, it takes about 9 hours to find the query plan for a single query. In comparison, our algorithms have very good scalability, and take less than 1 second.
even with 75 data sources. This experiment shows that the exhaustive algorithm is not practical for real systems that integrate 50 or more deep web sources, and need to support interactive queries.

**Evaluating Planning Algorithm for SPJ Queries**

We use 20 SPJ queries. For each of them, we generate query plan using each of the above three algorithms. We compare the three algorithms in two aspects: *query plan execution time* and *query plan quality*. In our case, the latter is simply evaluated by counting the number of data records returned in response to a query, since a better data source is expected to contain more information. The results are shown in Figure 5.10.

Figure 5.10(a) shows the *cumulative histogram* for the query execution time speedup of our planning algorithm over the steiner tree algorithm. The x-axis shows the speedup, and the y-axis shows the *cumulative percentage* of queries that have equal
or lesser speedup. In other words, in a cumulative histogram we present, a point \((x, y)\) denotes that \(y\%\) of the queries have a value less than or equal to \(x\). We observe that for about 60% of the SPJ queries, the query plans generated by our algorithm achieves a speedup over the plans from the steiner tree algorithm. Furthermore, for about one third of the queries, the execution speedup achieved is larger than 30%. This is because the Steiner algorithm requires pair-wise node connection, and therefore, it may have to include some unnecessary data sources. In terms of number of returned results, our algorithm and the steiner tree algorithm are very similar.

Figure 5.10(b) and Figure 5.10(c) show the comparison between our algorithm and the exhaustive search algorithm in terms of query plan execution time and query plan quality, respectively. From Figure 5.10(b), we have the following observations. First, for about 60% of the SPJ queries, the query plans from our algorithm have exactly the same execution time as the optimal plan generated by the exhaustive search algorithm. Second, for more than 80% of the queries, the execution slow down of the plans from our algorithm is no more than 20%. From Figure 5.10(c), we have similar observations. For about 60% of the queries, our plans have the same result quality as the optimal plans, and for over 90% of the queries, the result quality from our plans is no more than 20% lower than the optimal one. These results show although in some cases (less than 20% of the queries), our plans are not as good as the optimal plans, for a vast majority of the cases, our query plans are as efficient and effective as the optimal plans. Considering the previous results (Figure 5.9), where we show that lack of scalability of the optimal algorithms, these results establish that our method will be practical for real systems.
Figure 5.11: Query Planning Algorithms Comparison on Aggregation/Nested Queries

Evaluating Query Planning Algorithms for Aggregation and Nested Queries

In this experiment, we use 29 queries, including 16 aggregation queries and 13 queries with nested sub-queries. To the best of our knowledge, our work is the first one to systematically address the problem of query planning for aggregation and nested queries over the deep web. Therefore, we can compare our query planning algorithm with the exhaustive search algorithm only.

For each query, we compare the query execution time and results count between our planning algorithms and the exhaustive algorithm. We show the results in Figure 5.11. Figure 5.11(a) and Figure 5.11(c) shows the comparison results on query execution time and result counts, respectively. We have the following two observations. First (from Figure 5.11(a)), for over 85% of the aggregation queries, the execution time of the query plans generated by our method is close to that of the plans generated by the exhaustive method (slow down smaller than 20%). Second, (from Figure 5.11(c)), for over 90% of the queries, the result quality from our query
plans is very close to the result quality of optimal plans (result decrease less than
10%).

Figure 5.11(b) and Figure 5.11(d) shows the results on nested queries. For nested
queries, we could observe similar patterns as aggregation queries. Overall, our results
show that our query planning algorithms can generate plans with near-optimal effi-
ciency and results quality. In view of the low query planning time and high scalability,
this shows that our methods are practical for real deep web integration and querying
systems.

5.7.3 Evaluation of Optimization Methods

In this section, we evaluate the performance of the proposed optimization techniques.
In this experiment, we consider five different optimization schemes, which are NO (no
optimization), Merging (only perform plan merging presented in Section 5.4), Group-
ing (only perform the two grouping optimizations described in Section 5.5), M+G
(combine the optimizations from Section 5.4 and Section 5.5), and K-Optimization
(the existing optimization method proposed by Kementsietsidis et al [84]). We used
50 queries for this experiment. We execute the queries using all five schemes.

We focus on two metrics, which are the total number of data records transmitted
during the execution and the query execution time.

The results are shown in Figure 5.12. The x-axis in both sub-figures shows the
scheme used. The y-axis in sub-figure (a) shows the total number of data records
transmitted, whereas in sub-figure (b), it shows the speedup of the query execution
time normalized with respect to the NO version. The percentage above the bars in
sub-figure(a) shows the percentage of data records reduced using the corresponding
optimization scheme.
From the sub-figure (a), we have the following observations. 1). Using the merging and grouping optimizations independently, the total number of transmitted data records is decreased by about 25% and 20%, respectively. 2). When these two are combined (see the M+G scheme), we achieved a 40% decrease in the number of transmitted records. 3). Comparing our M+G scheme with the K-Optimization scheme, our combined optimization method reduces more than twice the number of data records compared with Kementzietsidis et al’s method. This shows the effectiveness of our optimization methods.

From sub-figure (b), we observe a similar trend. The execution speedups achieved using Merging and Grouping independently are 1.3 and 1.4, respectively. We achieved a 1.7 time speedup when we combined them. The execution speedup achieved using the Kementzietsidis et al’s method is somewhat lower than our Merging method, and much lower than our combined M+G method.
5.8 Summary

In this chapter, we have presented a framework for answering complex structured queries over the deep web. Currently, we have focused on three types of advance queries, SPJ queries, aggregation queries, and nested queries. We have formulated the query planning problem and developed new query planning algorithm for each of these types of queries. We have also described several optimization techniques to speedup query execution.

To demonstrate the efficiency of our algorithms, we compared our algorithms with the existing Steiner algorithm and an exhaustive search algorithm. We show the exhaustive algorithm scales very bad. For about 90% of the experimental queries, our planning algorithm could generate query plans which has nearly the same execution time and result quality as the optimal plans generated by the exhaustive algorithm. For about half of the SPJ queries, the execution time of the plans generated by our algorithm is significantly lower than the plans generated by the steiner tree algorithm. We also show that when compared with an existing query optimization method, our new optimization techniques have better performance, in terms of both reducing the number of transmitted data records, and the execution speedups.
CHAPTER 6
QUERY-PLAN-DRIVEN QUERY CACHING FOR
ACCELERATING SEARCH OVER DEEP WEB DATA SOURCES

Executing a query plan generated by the query planning algorithm introduced in Chapter 5 often involves a sequence of inter-dependent deep web data sources. Due to the load on server-side and the network delays, users can often experience high response time while querying the deep web. A recent study from a deep web integration system shows that nearly 80% of the execution time is spent on data delivery between the server and the clients [129]. Thus, improving the efficiency of deep web data searches is a major challenge.

6.1 Motivation

In this chapter, we propose a solution driven by the following observations. First, there is data redundancy across deep web data sources, i.e., the same data can be obtained from multiple sources [15]. Second, deep web data sources return query answers in an All-In-One fashion, i.e., values of all the attributes in the source’s output schema are returned, irrespective of the specific attributes requested in the query. For example, if we search for air tickets on Expedia.com, all flight-related data will be returned (flight number, aircraft model, price, etc), even if you only needed
to know about the price. Third, from a recent study on the surface web [121] and our experience with a specific deep web integration system [129], we found that users tend to issue related or similar queries over a short period of time.

The above specific features of deep web data sources motivate us to develop a novel query caching method. Our approach is distinct from the existing large volume of research on data caching in relational databases, as we not only cache previous extracted data, but more importantly, cache the query plans for previous queries. If query plans are cached and reused in a proper way, the possibility of data reuse can be significantly increased by taking advantage of data redundancy from related data sources.

The following motivating example further explains our idea.

**Example:** A biologist interested in SNP issues a query $Q_1=\{\text{Chromosome 10, NSYNSNP, Protein}_\text{Seq}\}$ first. This query aims to find all non-synonymous SNPs located in chromosome 10 and the sequences of the proteins encoded in this chromosome. The query plan for the query $Q_1$ is shown in sub-figure (a) of the Figure 6.1. This query plan shows that given a chromosome number, we prefer to use the SNP500Cancer3 data source to obtain the non-synonymous SNPs and the HP data source to obtain the protein sequences. A subsequent query $Q_2=\{\text{ERCC6, NSYN-SNP, ORTH}_\text{BLAST}\}$ aims to obtain all the non-synonymous SNPs located in the gene ERCC6 and the BLAST results of the proteins encoded by this gene. If we consider the query $Q_2$ in isolation from the query $Q_1$, the query plan for $Q_2$ we generate is as shown in sub-figure (b). In the query plan for query $Q_2$, given a gene name, we prefer to use dbSNP2 data source to obtain the non-synonymous SNPs in the gene and then use the Protein data source to obtain protein sequences. The SNPs returned from dbSNP2 will be the input into the dbSNP1 data source to obtain the amino acid position information. Finally, combining the amino acid positions from dbSNP1 and
the protein sequences from Protein as input to the BLAST data source, we can obtain the final result for ORTH_BLAST.

We can note that although the two sets of data sources, shown shaded in sub-figures (a) and (b), respectively, take different input attributes (SNPCancer3 and HP take chromosome as input, and dbSNP2 and Protein take gene name as input), they have data redundancy, i.e., we can obtain SNP information and protein sequence from both sets of data sources. Since the query plan of Q1 is cached, we want to process query Q2 in the fashion that uses the data sources in the plan of Q1. For this purpose, we need to add another data source, SNPCancer2, to find the chromosome in which ERCC6 is located. The detour query plan is shown in the sub-figure (c). The reused parts from sub-figure (a) and sub-figure (c) are enclosed by dotted rectangles. We can observe that SNPCancer2 takes gene name as input and returns the chromosome in which this gene is located. Then, the obtained chromosome can be used as input to the data sources in the query plan of Q1. The detour plan can increase the possibility of data reuse and speed up query execution if ERCC6 is located in chromosome 10 (which is true), because the cached data about SNP and proteins from Q1 can be reused. Thus, the execution time can be significantly reduced.

In this chapter, we propose a query-plan-driven query caching strategy. We not only cache previously extracted data, but also cache the earlier queries and their plans. In the context of the example above, the existing query caching techniques, which can be considered as being data-driven, will check for the overlap between the expected results from Q2 and the cached results from Q1. But in the example above, since Q1 does not involve genes, the cached data about Q1 will not indicate the information about the gene ERCC6, although this information is located in the chromosome 10. As a result, without caching earlier plans, certain reusable data, arising because of cross-source data redundancy, cannot be fully exploited. In our approach, reusing
previously cached plans can help enable greater reuse of the previously cached data, thus accelerating the response for the new query. It should be noted that because deep web queries are read-only queries and the content of such data sources is often relatively static, we do not consider concurrency control issues or advanced data caching techniques [82, 25, 136].

In the literature, many systems and algorithms have been proposed to address the problem of query optimization and data caching in the context of relational databases or SQL queries [54, 50, 73, 44, 115, 76, 117, 113, 38, 45]. Our approach differs from the existing work in the following ways. First, since data tables in relational databases are designed according to normal forms, cross-table data redundancy is avoided [42]. The possibility of data reuse is smaller than deep web data sources in which cross-source data redundancy exists. Second, relational databases only return the attribute values that are specified in the SQL Selection clause and other related attributes from the same data row are hidden from the user. But the All-In-One nature of deep web
data sources increases the potential of data reuse. Third, the keyword search queries we are considering are different from the SQL queries. The SQL join and aggregation operations, which are often the main focus in query caching techniques for relational database, do not arise in keyword queries. Finally, in the existing work, the purpose of comparing the new query with the previously cached queries is to determine what cached data can be reused. But in our approach, as we had stated above, we find similar earlier queries in order to use their query plans, and not only for finding the reusable data.

6.2 Query Reuse Problem Formulation

This section gives a formal description of the problem we are focusing on. Initially, we define the query and query plans we consider in our work.

6.2.1 Queries and Query Plans

Throughout this chapter, the query reuse problem and algorithms are presented based on the simple ordinary keyword structure query introduced in Chapter 5. The algorithms proposed in this chapter can be easily applied to the other two types of queries, simple aggregation query and nested entity query, with minor modifications. 

**Query Q:** Q comprises n, n > 1, search terms, and it is formally denoted as \( Q = \{t_1, t_2, \ldots, t_n\} \). Among these n search terms, there is at least one search term, \( t_i \), which is a concrete entity name. Such a term helps initiate answering of the query. For example, \( Q_1 \) in Section 1 has three search terms, \{t1=Chromosome10, t2=NSYNSNP, t3=Protein_Seq\} and, here, \( t_1 \) is the entity name.

**Query Plan P:** A query plan \( P \) of a query \( Q \) is defined as a graph \( P = (V, E, V_0) \). \( V \) is a finite set of data sources involved in the query plan. \( E \) is a set of directed edges in the plan, each of which indicating an inter-dependence relation between a
pair of data sources. If a directed edge $e$ is from the node $A$ to the node $B$, it implies that the data source $A$ provides the necessary input attributes for the data source $B$. $V_0$ is the set of starting data sources of the plan. The starting data sources take the entity name term(s) in $Q$ as the input to initiate the query plan.

As an example, for the query plan in sub-figure (c) of Figure 6.1, we have $V=\{\text{SNPCancer2, SNPCancer3, HP, BLAST}\}$ and $V_0=\{\text{SNPCancer2}\}$. $\text{SNPCancer2}$ takes gene ERCC6 as input to initiate the query plan. The SNP information (corresponding to search term $\text{NSYNSNP}$) is covered by $\text{SNPCancer3}$ and the BLAST information (corresponding to search term $\text{ORTH.BLAST}$) is covered by the $\text{BLAST}$ data source. The edge between $\text{SNPCancer3}$ and $\text{BLAST}$ in sub-figure (c) shows that the output from $\text{SNPCancer3}$, which are SNP and amino acid position, can be used as the input of $\text{BLAST}$.

6.2.2 Query Reuse Problem

To formulate the query reuse problem, we use the following definitions.

**Query Subplan:** We define a query subplan $SubP$ of an original query plan $P$ as a connected sub-graph of the original query plan graph. Formally, $SubP = (V', E', V'_0)$ where $V' \subseteq V$, $E' \subseteq E$, $V'_0 \subseteq V$ and $|V'_0| > 0$.

**Query Subplan Set:** The query subplan set $SubPSet$ of query plan $P$ is the set containing all possible query subplans of $P$.

**$\Psi$ Selection:** Given a new query and a previous query with plan $P$, we want to determine the subplan of $P$ that, among all subplans of $P$, will enable maximal reuse for the new query, and of all subplans enabling such maximal reuse, will involve fewest
data sources. This is captured through the $\Psi$ selection operator. The selected query subplan is denoted as $\Psi(SubPSet(P))$.

**Remainder Query:** The remainder query $RQ$ of a query $Q$ contains the keywords in $Q$ which cannot be covered by using previous query plans.

**Problem Definition:** Using the above terms, our query caching problem is formally stated as follows. We are given a list of $n$ previous issued keyword queries, each of which has an associated query plan $P_i$. Given a new query, we want to construct its query plan using a list of $\Psi$ selected query subplans,

$$\Psi(SubPSet(P_1)), \Psi(SubPSet(P_2)), \ldots, \Psi(SubPSet(P_n))$$

and other necessary data sources. We obtain the query plan for the new query, and while executing this query plan, we reuse previously cached data.

Note that the quality of the query plan thus obtained may not necessarily be as high as the one we could generate independently. However, a query plan based on query plans of previously executed queries is likely to enable more data reuse, and therefore, accelerate the execution of the queries.

**6.2.3 Solution Overview**

To solve the problem formulated as above, we take the following steps.

1. We define a *reusability* metric to be able to identify the query plans that we would be beneficial to reuse. We consider these as *reusable* queries and query plans.

2. We select a list of such reusable previous queries and their plans, and then apply the $\Psi$ selection function to obtain the sub-query plans we will like to reuse. Here, one challenge that we address is performing $\Psi$ selection without explicit enumeration of all query subplans.
3. We modify the bidirectional query planning algorithm proposed in chapter 5 so as to generate a query plan based on a list of selected reusable sub-query plans. Here, a particular issue is balancing the trade-off between reuse and plan quality.

The next two sections address the above three issues. The first two are described in Section 6.3 and the last one is detailed in Section 6.4.

6.3 Reusable Sub-Query Plan Selection

In this section, we describe how reusable query subplans are selected from cached queries and their plans. First, we state a reusability metric to capture the likelihood that a cached query can be reused for the new query. Next, we show that given a new query, the problem of selecting a subplan from the original plan, which enables maximal reuse, while involving the fewest data sources, is an NP-hard problem. We then propose a heuristic polynomial time algorithm, the \( \Psi \) selection algorithm, to select such a query subplan. Finally, we propose a greedy algorithm to select a list of reusable sub-query plans for a new query.

6.3.1 Query Reusability Metric

Suppose we have two queries \( Q_1 \) and \( Q_2 \), each of which comprises a number of keywords. We want to know the potential of reusing \( Q_1 \) in answering \( Q_2 \). The most intuitive method is to measure the similarity between \( Q_1 \) and \( Q_2 \), specifically, how many common terms they have. However, this simple method is not sufficient, as we can see from the following example. Given a pair of cities, a travel data source \( D \) returns five search terms, which are \( t_1=\text{ticket price}, t_2=\text{departure time}, t_3=\text{arrival time}, t_4=\text{aircraft model}, \) and \( t_5=\text{meal description} \). A traveler using this web-site only
wants to know the price, and the departure and arrival times, and thus, his query $Q_1$ would only contain three search terms, $t_1$, $t_2$, and $t_3$. Another user, who is a manager from a competing airline, is only concerned about the type of aircrafts used and the meals served on the flights. Thus, her query $Q_2$ would contain two search terms, $t_4$ and $t_5$. However, due to the All-In-One feature of deep web data sources, when the first query is being answered for a pair of cities $C_1$ and $C_2$, $t_4$ and $t_5$ are also returned. Thus, $Q_2$ can be answered completely by reusing the results of $Q_1$.

To capture this observation in the query reusability metric, we formally define an augmented query as follows.

**Augmented Query:** Suppose we have a query plan $P$ for a query $Q$. For each $v_i \in V$, we extract all terms $t_{ij} \in \text{Output}(v_i)$ and obtain a new query

$$Q^* = \{t_{11}, t_{12}, \ldots, t_{1k_1}, \ldots, t_{|V|1}, \ldots, t_{|V|k_{|V|}}\}$$

where $k_i = |\text{Output}(v_i)|$. Clearly $Q^*$ is a superset of $Q$, and we say that $Q^*$ is an augmented query of $Q$ related to the plan $P$, and denote it as $A_P(Q)$. Considering the above example, since the travel data source $D$ is the query plan for answering $Q_1 = \{\text{price, departure time, arrival time}\}$, the augmented query of $Q_1$ is $A_P(Q_1) = \{\text{price, departure time, arrival time, aircraft model, meal}\}$.

To compute the reusability of $Q_1$, for which we had generated the plan $P$, for answering $Q_2$, we first obtain the augmented query of $Q_1$, denoted as $A_P(Q_1)$. Our query similarity metric computes the similarity between $A_P(Q_1)$ and $Q_2$. Specifically, we use $S_{10}$ to denote the number of search terms that $Q_2$ contains but $A_P(Q_1)$ does not, $S_{01}$ to denote the number of search terms that $A_P(Q_1)$ contains but $Q_2$ does not, and $S_{11}$ to denote the number of search terms that both $Q_2$ and $A_P(Q_1)$ contain. If a search term $t$ is contained in both $Q_2$ and $A_P(Q_1)$, it can be reused in $Q_2$. In the above example, the search term **aircraft model** and **meal description** can be reused in $Q_2$. Thus, $S_{11}$ should be considered as the **benefit for reuse**. $S_{10}$, which
denotes the number of search terms that are requested by $Q_2$ but not retrieved by $A_P(Q_1)$, should be considered as a penalty for reuse. Search terms which are retrieved by $A_P(Q_1)$ but not requested by $Q_2$ lower the similarity, even though they do not adversely impact the reuse of $Q_1$ for $Q_2$. Thus, $S_{01}$ is considered as partial penalty for reuse.

Formally the similarity function between $Q_1$ and $Q_2$ is defined as follows:

$$Sim(Q_1, Q_2) = \frac{S_{11} - S_{10} - \frac{S_{01}}{S_{11} + 1}}{S_{11} + S_{10} + S_{01}}$$

The numerator captures the pure benefit for reuse by subtracting the penalty and partial penalty from the benefit for reuse. To reflect $S_{01}$ is only a partial penalty, we lower the weight of $S_{01}$ by a factor of $S_{11} + 1$, which is to emphasize the ratio between $S_{01}$ and $S_{11}$. A larger ratio suggests that $Q_1$ contains many unrelated terms to $Q_2$, which gives a higher weight on the partial penalty and a smaller ratio suggests otherwise. The denominator is a normalization factor.

The value of the above function ranges from $-1$ to $1$. We convert it to range from $0$ to $1$, by modifying it as follows:

$$Sim(Q_1, Q_2) = \frac{2S_{11} + (1 - \frac{1}{S_{11} + 1})S_{01}}{2(S_{11} + S_{10} + S_{01})}$$

6.3.2 Ψ Selection of Reusable Query Subplan

Suppose for a new query $Q_2$, we have selected an earlier query $Q_1$, which has a high reusability value. However, we cannot just reuse the entire plan of $Q_1$, because the plan of $Q_1$ may extract some terms which are not requested by $Q_2$. As a result, we need to use a selection function to find a subplan that satisfies the following two conditions: 1) the subplan maximally covers $Q_2$, and 2) the size of the subplan is minimal. We use the maximal coverage condition to select the previous subplan which could enable the maximal reuse. For the second condition, when we have a cache hit,
Figure 6.2: Example for $\Psi$ Selection: (a) Query Plan of Q1, (b) $\Psi$ Selection Step 1, (c) $\Psi$ Selection Step 2, (d) $\Psi$ Selection Step 3 (minimal size obtained) (e) A query plan with only partial coverage.

the cost would be the number of local disk accesses. This, in turn, is proportional to the number of data sources in the reused subplans. As a result, the minimal size condition is used to select the previous subplan which causes the lowest cost.

To explain the idea, we use the following example. The augmented query of Q1 is $A_P(Q1) = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$. The query plan of Q1 is shown in the sub-figure (a) of Figure 6.2. The terms inside each nodes indicate the output set of the data source, and the terms located above the arrow of each link indicate the input terms of the pointed data source. For example, the data source $A$ has the output $t_1$ and $t_2$. The input set of the source $F$ is $t_4$ and $t_5$, which are provided by the sources $D$ and $E$, respectively. The new query we consider is $Q2 = \{t_1, t_3, t_4\}$.

Maximal Coverage Query Subplan: We consider a query subplan $SubP$ to be maximally covering a new query $Q2$ if and only if the keywords covered by all the data sources in $SubP$ is a superset of the common terms in $Q2$ and $A_P(Q1)$. Formally, 

\[ \bigcup_{D_i \in SubP} Output(D_i) \supseteq Q2 \cap A_P(Q1). \]
In the example in Figure 6.2, sub-figures (b),(c) and (d) are all *Maximal Coverage Query Subplans* of the plan (a) w.r.t $Q_2$, but the subplan in sub-figure (e) is not, because it does not contain term $t_4$. The subplan in sub-figure (d) shows *maximal coverage* subplans with *minimal size*.

**Subplan Selection Problem Formulation:**

The subplan selection problem can be converted to a graph problem as follows. We are given a graph $G = (V, E)$, where $G$ represents the query plan of a previous query, $V$ represents a set of nodes (data sources), and $E$ represents a set of edges. Each $v_i \in V$ covers a set of terms or element $S_i = \{s_1, s_2, \ldots, s_k\}, k \geq 1$. We define an *entire graph set* $GS$ to be the union of all sets of elements covered by each $v_i \in V$, denoted as $GS = \bigcup_{v_i \in V} S_i$. Given a new set $T$, which is a subset of $GS$, we want to find a *connected subgraph* of $G$, $SubG$, which covers all elements in $T$, while having the *minimal size*. We call this problem the *connected subgraph set cover* problem.

The *connected subgraph set cover* problem seems like the standard *set cover* problem, but with the following distinction: we not only need to cover all elements in a set, but also need to make sure that the selected sets (nodes) form a *connected* subgraph.

Lemma 6.3.1 shows that the *connected subgraph set cover* problem is an NP-hard problem. This lemma can be proved from a reduction from the set cover problem.

**Lemma 6.3.1.** The *connected subgraph set cover* problem is NP-hard.

**Proof.** The standard *set cover* NP-hard problem can be reduced to the *connected subgraph set cover* problem. We will describe how to construct a graph from any instance of the set cover problem. Further, we will show how, in this graph, any optimal solution for the set cover problem corresponds to an optimal solution to the connected subgraph set cover problem.

Obviously, given a subgraph of a graph $G$, we can check whether this subgraph
covers a set in polynomial time. This shows that our problem belongs to NP. Next we will show it is also NP-hard.

Let \( C = \{C_1, C_2, \ldots, C_m\} \), be an instance of the set cover problem. Recall that a solution to the set cover problem is a minimum subset \( C' \subseteq C \), such that each element of \( S = \bigcup_{i=1}^{m} C_i \) is contained in \( C' \). Denoting the elements of \( S \) by \( s_1, s_2, \ldots, s_n \), the DAG \( G = (V, E) \) is constructed as follows. The node set \( V \) of \( G \) is given by

\[
V = \{s_1, \ldots, s_n, C_1, \ldots, C_m, W\}
\]

Here, \( C_i \) is a set in the set cover problem, and \( W \) is a new node added to \( G \). Let each \( s_i \) node in \( G \) cover the element \( s_i \), and let each \( C_j \) node covers nothing and the node \( W \) covers a new element we introduce, which we denote as \( s^* \). The given set \( T \) is defined as \( \{s_1, \ldots, s_n, s^*\} \).

\( G \) contains two sets of edges. First, for each of the \( C_i \) nodes, there is a directed edge connecting the \( C_i \) to the \( W \) node. Second, each \( s_i \) node is connected with any node \( C_j \), for which \( s_i \in C_j \). Obviously, the graph \( G \) can be constructed in polynomial time. Figure 6.3 shows an example for the set cover instance:

\[
\{\{s_1, s_3\}, \{s_1\}, \{s_3\}, \{s_2, s_3\}\}
\]

Observe that any connected subgraph of \( G \) that covers all elements in \( T = \{s_1, \ldots, s_n, s^*\} \), must contain all \( s_i \) nodes as well as the \( W \) node. Given the nature of the graph, this will only be possible through the inclusion of some \( C_j \) nodes (without the inclusion of any \( C_j \) node, the subgraph will not be connected). A critical observation is that in order to make the size of the subgraph minimal, we must use the minimal number of \( C_j \) nodes. But, now, this is equivalent to minimizing the set cover represented by \( C_j \). This shows that any set of \( C_j \) nodes that are contained in the minimal connected subgraph of \( G \), such that it covers all elements in \( T \), is a minimal set cover also, and vice-versa. In Figure 6.3, the shaded subgraph shows
an instance of the minimal connected subgraph covering $T$. At the same time, the shaded $C_j$ nodes form a minimal set cover.

In view of the above lemma, we have developed a polynomial-time but heuristic $\Psi$ selection algorithm. The algorithm is based on two heuristics, The first heuristic is to give preference to nodes covering search terms, which implies that we prefer to include the node which can cover some search terms of the new query in the final selected subplan. The second heuristic is to give preference to nodes close to each other. The idea of this heuristic is shown in Figure 6.4. In this example, the set of elements we want to cover is $T = \{s_1, s_2, s_3\}$. We can see that the nodes $A$ and $H$ cover $T$ with minimal size, which is 2. But, if we want to connect $A$ and $H$ using a subgraph, the smallest subgraph has a size of 4, (it will contain $A$, $B$, $E$, and $H$). We can also observe that the nodes $C$, $D$, and $F$ also cover $T$. Although here we use three nodes to cover $T$, but the subgraph connecting these three nodes only has a size of 3. The reason is that the nodes we selected were already connected to each other.
The Ψ selection algorithm scans the data sources in the query plan, in the reverse
topological order. During the topological scan, a data source \( D \) in the plan \( P \) of
previous query \( Q_1 \) is removed if it meets any of the following two conditions: 1) \( D \)
cannot provide any terms requested by \( Q_2 \) (heuristic 1) 2) \( D \) can provide some terms
requested in \( Q_2 \), but is subsumed by one of its predecessors (heuristic 2).

To explain what we mean by subsumed, we use the following definitions. We say
that the data source \( D_i \) contains \( D_j \) according to a new query \( Q_2 \) if and only if
\( \text{Output}(D_i) \supseteq \text{Output}(D_j) \cap Q_2 \). We say that the data source \( D_i \) topologically
contains \( D_j \) if and only if \( D_i \) topologically precedes \( D_j \) and \( D_i \) contains \( D_j \).
Finally, we say that the data source \( D_i \) subsumes \( D_j \) if the following two conditions
hold: 1) \( D_i \) topologically contains \( D_j \), and 2) \( D_j \) has no descendants in the current
subplan or although \( D_j \) has descendants, the output from \( D_i \) could provide the input
of \( D_j \)’s descendants. When \( D_i \) subsumes \( D_j \), we can remove \( D_j \), and if \( D_j \) has
descendants, we can link \( D_i \) directly to \( D_j \)’s descendants. From the above definition,
we can know that if \( D_i \) subsumes \( D_j \), \( D_i \) covers the required search terms that \( D_j \) can
and meanwhile \( D_i \) topologically proceeds \( D_j \). Since we scan data sources in the reverse
topological order, if we find other data sources which covers other requested search

\begin{figure}
\centering
\includegraphics[width=0.8\textwidth]{figure6.4.png}
\caption{An Example Illustrating the Heuristic 2}
\end{figure}
terms later, these data sources would be nearer to $D_i$ than $D_j$. The replacement of $D_j$ for $D_i$ shows the application of the second heuristic. The $\Psi$ selection method is shown in Algorithm 6.3.1.

We use the example in Figure 6.2 to illustrate the $\Psi$ selection method. The algorithm starts from the first data source in the reverse topological order of the query plan, which is $F$ in sub-figure(a). Since $F$ cannot provide any terms in $Q_2$, which are $t_1$, $t_3$, and $t_4$, $F$ is removed from the plan, resulting in the subplan in sub-figure (b). Then, we continue with the data sources at the next topological level, which are $D$, $E$, and $G$. Similar to $F$, $E$ and $G$ cannot provide any terms in $Q_2$, so they are removed. However, $D$ provides term $t_4$, which is requested by $Q_2$. So, $D$ is kept in the plan. The subplan in sub-figure(c) shows the subplan generated after the second step. Data sources $B$ and $C$ are examined in the third step. $B$ and $C$ both provide terms in $Q_2$, and we observe that $B$ subsumes $D$ due to the following reasons: 1) $t_4$, which is provided by $D$ and requested by $Q_2$, is also provided by $B$ ($B$ contains $D$), 2) $B$ topologically proceeds $D$ ($B$ topologically contains $D$), and 3) $D$ doesn’t have dependents. As a result, $D$ is subsumed by $B$ and $D$ is removed from the plan. The subplan in sub-figure(d) shows the final subplan generated by the $\Psi$ selection method.


\section*{Algorithm 6.3.1: \(\Psi\)-Selection\((P, Q2)\)}

\begin{itemize}
  \item sort the data sources in \(P\) in the reverse topological order
  \item \textbf{foreach} topological level in the sorted order
    \begin{itemize}
      \item extract the data sources in the current topological level
      \item put them into a set \(SD\)
      \item \textbf{foreach} \(D_i \in SD\)
        \begin{itemize}
          \item \textbf{if} \(Output(D_i) \cap Q2 = \emptyset\)
            \begin{itemize}
              \item remove \(D_i\) from \(P\)
            \end{itemize}
          \end{itemize}
        \end{itemize}
    \end{itemize}
  \end{itemize}

\subsection{6.3.3 Algorithm for Selecting Reusable Query Subplans}

Our overall algorithm combines the ideas from the discussion above. One issue, however, is that for certain queries, we may want to reuse subplans from multiple prior queries. This is done through a greedy algorithm as follows.

**Step 1:** Given a new query \(Q2\), we scan all previous cached queries, and find the query \(Q1\), which has the highest reusability value.

**Step 2:** If the reusability value is greater than a threshold, the algorithm continues. Otherwise, we know that no additional query is reusable, and the algorithm terminates.

**Step 3:** We obtain the augmented query of \(Q1\) and invoke the \(\Psi\) selection function
to obtain the corresponding *reusable query subplan* and add it into a list of selected reusable query subplans.

**Step 4:** We update the query $Q_2$ by deleting all keywords which are covered by the reusable query subplan generated in the previous step.

**Step 5:** We repeat steps 1 to 4 using the updated query $Q_2$, until the algorithm terminates. The last updated $Q_2$ is the *remainder query*.

### 6.4 Query Plan Generation Algorithm QPCache

In the previous section, we described how we convert a new query $Q_2$ into a list of reusable query subplans and a *remainder* query. Now, our goal is to generate a query plan for $Q_2$. The planning algorithm QPCache we present for this purpose is modified from the bidirectional deep web query planning algorithm proposed in Chapter 5.

#### 6.4.1 Modified Algorithm for Enabling Reuse

We first give the intuition behind query planning using previous plans, and then we will explain the detail of two modifications we made to the original bidirectional query planning algorithm. We then illustrate our algorithm with an example.

The query planning algorithm QPCache takes two input parameters. The first is the remainder query, denoted as $RQ = \{t_1, \ldots, t_k\}, k \geq 0$, where each $t_i$ is a keyword, and the second parameter is the query template $QT = \{qt_1, \ldots, qt_n\}, n \geq 0$, where each $qt_i$ is a reusable query subplan. If $n = 0$, there is no reusable query template, so we simply invoke the original bidirectional planning algorithm. For other cases, we treat the remainder query as an ordinary query and do the query planning using the bidirectional algorithm. But we want to use one of the query templates whenever possible. To achieve this, we modify the original algorithm as follows.

PQCache algorithm is initialized similar to the bidirectional planning algorithm.
The only difference is that we increase the scores of the data source nodes which are involved in at least one query template. The reason is that we want to give higher priority for the nodes which are in the query templates so that the query templates can be explored earlier.

We perform the forward and backward exploration in the normal fashion, till we reach a point where the current selected data source to be explored, denoted as \( CN \), is in one of the templates in \( QT, qt_i \). Now, we suspend the normal bidirectional planning, and instead, we do a depth first exploration from the node \( CN \), along the edges in \( qt_i \), to include all data sources in \( qt_i \) to be in the query plan of \( RQ \). After the exploration of data sources in \( qt_i \), we return to the normal bidirectional planning. The depth-first exploration of the query template is referred to as a *detour* exploration.

**Detour Exploration:** Suppose the data source \( CN \) is currently being explored, and \( CN \) is involved in a query template \( qt \). Procedure *Detour* in Algorithm 6.4.1 shows the *detour exploration* method. In the detour algorithm, we first do a depth-first backward exploration from the node \( CN \) till we reach the starting node(s) of the query template \( qt \). After the backward exploration, we start from node \( CN \) again to do a depth-first forward exploration.

The reason we do backward exploration on the query template first is to make sure that when a node is being explored, all its immediate parents have already been explored.

Procedures *BackwardDetour* and *ForwardDetour* in Algorithm 6.4.1 show the details of detour exploration. In the backward detour function, all the immediate parents of the current node are explored. The backward detour function is a recursive function which performs backward explorations on the parents. The forward detour function works similarly and the immediate descendants of the current node are explored in a recursive fashion. An important difference between backward detour and
forward detour exploration is that in the backward version, if we reach the starting node(s) of the query templates, we do not just stop. Instead, we do an extra backward exploration from the starting node(s) of the query template to its parents outside the query template. The reason is that the query template is only a detour, and we ultimately need to connect the starting node(s) of the detour with node(s) in the main query path that is generated by the original bidirectional algorithm.
Algorithm 6.4.1: QPCache\((QT,RQ)\)

\[\text{Detour}(QT,CN)\]
- Backward-Detour\((QT,CN)\), Forward-Detour\((QT,CN)\)

\[\text{END}\]

\[\text{Backward-Detour}(QT,CN)\]
- if \(\text{Parents}(CN) = \Phi\)
  - Bidirectional-Backward-Explore\((CN)\)
- else
  - foreach \(U \in \text{Parents}(CN)\)
    - Modified-Edge-Explore\((U,CN)\), Backward-Detour\((QT,U)\)

\[\text{END}\]

\[\text{Forward-Detour}(QT,CN)\]
- foreach \(v \in \text{Children}(CN)\)
  - Modified-Edge-Explore\((CN,v)\), Forward-Detour\((QT,v)\)

\[\text{END}\]

\[\text{QPCache}(QT,RQ)\]
- Increase the scores of the nodes in query templates
- while \((\text{FQ} \neq \Phi \text{ or } \text{BQ} \neq \Phi)\)
  - select the node with the highest score from \(\text{FQ}\) and \(\text{BQ}\), call it \(CN\)
  - if \(CN \in QT\)
    - Detour\((QT,CN)\)
  - else
    - Bidirectional-Planning\((CN)\)

\[\text{END}\]

\textbf{Modified Edge Exploration:} In the original bidirectional planning algorithm,
edge exploration is conducted in two directions. In forward exploration, all out-going neighbors of the current node will be explored. In backward manner, all in-coming parents of the current node will be explored. The edge exploration function updates the shortest distance from a parent node to a children node and further propagates the distance to the parent node’s ancestors like the Dijkstra’s shortest path algorithm. To accommodate for the changes in QPCache algorithm, the edge exploration is also modified in the following ways. In the edge exploration function of the bidirectional planning algorithm, if the distance between the starting node(s) and a keyword via the current being explored node is shorter than the previous distance, the shortest distance is updated. But in the QPCache algorithm, because we want to give higher priority to the detour path which normally has a lower score, we lock the shortest distance if it is achieved by going through a detour path. This ensures that it is not updated by a path generated by the normal bidirectional algorithm. We call this shortest path locking for detour. The other issue is that although we give priority to the detour path, we do not want to penalize the normal path severely. In other words, we do not want to reuse previous plans and save some execution time, if it results in query plans with very low quality scores. As a result, the lock is released if the distance coming from a normal path is much shorter than the locked shortest distance obtaining by a detour path. The release of the lock is controlled by a specified threshold.

6.4.2 An Example for QPCache Algorithm

Figure 6.5 shows an example to illustrate the algorithm. There are two query templates with four and two data sources, respectively. The shaded nodes and thick dotted lines with arrows are used to highlight the data sources or edges newly inserted into the query plan. Suppose the two templates and the remainder query are
the input parameters of the QPCache algorithm. First, suppose the data source A is selected to be explored according to the remainder query. In the second step, one of the neighbors of A, C is explored and added to the query plan. Now we notice that C is in the query template 1, and we suspend the normal bidirectional algorithm and invoke the detour exploration method. Starting from the node C in query template 1, we first do a backward detour exploration to add B in template 1 to the plan as shown in the step 3. Because B is the starting node of template 1, we try to connect the detour path with the outside main path, so that we continue to do backward exploration from B to any its parents. In the step 4, B is connected with A. Now we return to template 1 and begin to do forward detour exploration from C. Since the nodes C and D form a hyper-parent for E, nodes D and E are both explored in this forward detour exploration as shown in step 5. Now, we are on the node E, and we note that the node E is involved in the query template 2. As a result, we continue the execution of detour exploration on the template 2. Since E is the starting node of template 2, and E is already connected with the main path, the backward detour
exploration for $E$ is skipped. We just do the forward detour exploration as shown in the step 6 and the node $F$ is added to the plan. Now the detour exploration is completed, and the control is returned back to the normal bidirectional algorithm. Recall that the bidirectional algorithm is suspended while traversing the neighbors of $A$. Suppose the next neighbor of $A$ being explored is $G$ and $G$ contains some keyword in the remainder query. Then, $G$ is added to the plan as shown in the step 7. Now suppose all keywords in the remainder query are covered. The $\text{QPCache}$ algorithm then terminates.

From the above example, we can see that the keywords in the remainder query is covered by node $A$ and $G$, and the remaining nodes cover the keywords in the original query (excluding the keywords from the remainder query).

### 6.4.3 Managing Cached Queries and Query Plans

We now briefly describe the heuristics we use for determining when a new query should be cached, and when a cached query should be evicted from the cache.

The admission of a new query plan depends on the newness of the plan. We define a node in the new query plan as a New Node if it is not in any reused query templates in the cache. An edge is a New Edge if at least one of the edge’s end points is a New Node. The Newness of the plan is the score coming from all new nodes and new edges as computed by the ranking function. The score coming from the remaining nodes and edges is defined as the Re-usage of the new plan. For a new query plan $np$, if $\text{Newness}(np) \geq \alpha \times Re - usage(np)$, $np$ is admitted.

When a new plan is admitted, it is likely that some previous cached plans are covered by the new plan, and should be replaced. The replacement criteria is based on a similarity score between the augmented query of the new query and the augmented
query of each previous cached query. If the similarity is greater than a threshold $\beta$, the previous cached query is removed.

6.5 Experimental Results

In this section, we describe the experiments we conducted to evaluate our techniques.

6.5.1 Experiment Setup

Our evaluation was done using 12 biological deep web data sources we had integrated in our previous work, which includes dbSNP, Entrez Gene, Protein, BLAST, SNP500, Seattle, SIFT, BIND, Human Protein, HGNC, Mouse SNP\(^1\), and ALFRED. We use an Oracle database to cache previous plans and results. A collaborating biologist provided us 24 real queries, which were divided into 4 groups, with each group intended to simulate a list of similar queries issued by a user in a session. The first

\(^1\)http://mousesnp.roche.com/
Figure 6.6: Comparison of Query Execution Time: (a) Group 1 Queries; (b) Group 2 Queries; (c) Group 3 Queries; (d) Group 4 Queries.
query in the first, third, and the fourth group, and the first two queries in the second
group, are considered seeds, and the rest of the queries in each group are considered
as new coming queries. The reason we put 2 seeds in the second group is that we
want to test our algorithm when there is a potential for reusing multiple previous
queries for one newly issued query.

Table 6.1 shows the Cumulative Reusability Score (CRS) for the 24 queries, which
is computed as follows. We consider all previous queries in the same group as one
single existing query, and then compute the reusability score between the new query
and the existing query. In our evaluation, we compared three scenarios. They are
No Caching (NC), Data Driven Caching (DDC), and Plan Driven Caching (PDC).
NC implies that we do not cache anything, and the query plan is generated using
the original query planning algorithm. DDC refers to the case when we use ordinary
data caching method in relational databases. Given a new query, a query plan is
generated using the original query planning algorithm without considering the special
data redundancy across deep web data sources. Then we check for overlap between
the new plan and cached plans, and we reuse cached data if it is possible. PDC
refers to the strategy proposed in this paper. The new query plan is generated by
reusing cached plans, and when the plan is executed, we reuse cached data if possible.
Overall, by comparing PDC against the other two schemes, we want to show that a
query-plan-driven query caching scheme is more effective than a data-driven query
caching scheme, or no caching.

6.5.2 Evaluation Metrics

Query Execution Time: Query Execution Time is the time for query planning and
the time for actually issuing the query on each of the deep web data sources included
in the plan.
**Query Plan Score:** For each query, we compute the score of the query plan generated by NC, DDC and PDC approaches. The score of the query plan is computed by the ranking function in our bidirectional query planning algorithm. We want to compare the quality of the plan generated by the three methods, and to examine whether the desired trade-off between the plan quality and the plan execution time is achieved.

**Actual Query Result:** For the query plans generated using Data Driven caching (DDC) and Plan Driven Caching (PDC) for all experimental queries, we record the final query results. We want to know whether the results from PDC are correct and complete comparing with the results from DDC. This metric can also be used to examine whether the desired trade-off between the plan quality and the plan execution time is achieved.

**Query Planning Time:** The query planning time for the three scenarios are recorded. We want to see how much query planning overhead is incurred while generating query plans using cached queries and plans, where we perform detour exploration.

### 6.5.3 Experimental Results

**Comparing Query Execution Times:** Figure 6.6 shows the comparison of query execution time for the three scenarios, NC, DDC, and PDC. The x axis is the query number in each group (the seed is excluded), and the y axis is the speedup that each scenario achieves compared to the baseline, which is NC. Because of this normalization, the values for the NC case is always 1.

In the figure, five queries are highlighted by ovals, and will be discussed below. For all the remaining 14 queries, both DDC and PDC significantly outperformed NC. PDC outperformed DDC in 13 of these 14 queries, and in 8 of these 14 queries, PDC
achieved at least twice the speedup of DDC, showing the benefits of reusing cached query plans.

For the five queries highlighted by ovals, we can see that the two reused versions do not obtain any speedups over the NC version. The reason is that for these five queries, our QPCache algorithm decides not to reuse any previous cached results, because reusing cached results is likely to give a query plan with very low quality score. The overhead of checking local databases and QPCache algorithm causes DDC and PDC having lower performance than NC.

We also sort the queries in each group by their CRS score and we observe that there is a positive, but not very high, correlation between the speedup obtained by PDC and the CRS score. The reason is that the speedup of a query is more dependent on the response time of the data sources which are reused in the query plan, and not the number of data sources that can be reused.

**Comparing Query Plan Scores:** In Section 6.4 we mentioned that our QPCache algorithm gives priority to reused query plans, and as a result, may generate query plans with a relatively low plan quality score. But if the algorithm detects that the score of the query plan generated by reusing is likely to be too low, the algorithm
reduces to the normal bidirectional planning algorithm. We now examine if our algorithm achieves the desired trade-off between the execution speedup and the plan quality score.

For each query, we first record the quality score of the query plan if it was generated with PDC. This is denote it as $\text{Score}_{PDC}$. Then, we also record the quality score of the query plan if generated with NC, and denote it as $\text{Score}_{NC}$. The ratio of them are shown in Figure 6.7. The $x$ axis is the 19 queries (seeds excluded), and the $y$ axis is the ratio $\frac{\text{Score}_{NC}}{\text{Score}_{PDC}}$. The reference line is $\text{Ratio} = 2$, which means that the lowest score we accept for the query plan generated with reuse is half of the quality score of the plan generated by the original planning algorithm.

From the figure, we observe that except for six queries, the ratios are below the reference line, though above 1. This shows that the query plans generated by reusing previous plans indeed have lower quality score, but the scores are not very low in most cases, and as we have shown earlier, they still result in overall speedup. For the six queries for which the values are above the reference line, we can observe that except for query number 13 (rectangle), all other five queries (circle) are the ones we highlighted by ovals in Figure 6.6. This shows that when the quality score of the plan generated by reusing tends to be very low, our algorithm correctly decides to generate the plan from the original method.

**Cache Status During Experiments:** Figure 6.8 shows the status of the query cache in our system during our experiments. Each row illustrates the change of the query cache for all queries in one group. The numbers in each box show the query plans which are in the query cache after the execution of the current query. For example, the third box in the second row from the left shows that after the execution of query 2.1, there are three query plans cached, which correspond to the seed B, the seed C, and the query plan of the query 2.1, respectively. If the number of a cached
query plan is in bold italic font, it means that this cached query plan is reused for the current query. If the number of a cached query plan is inside a rectangle with dotted border, it means that this cached query plan is just added to the query cache after the execution of the current query. If the number of a cached query plan is shaded, it implies that this cached query plan is deleted from the cache after the execution of the current query. For example, the third box in the third row from the left shows that the query plan of seed D is reused for the current query 3.2, and after the execution of query 3.2, the cached plan of seed D is deleted from the cache and the plan of query 3.2 is added to the cache.

The query cache status shows that our algorithm does not only reuse the seed for each group. The algorithm does select the appropriate cached query to reuse and the query cache is updated by our admission and replacement algorithm. From the last boxes in groups 2 and 3, and the last three boxes in group 4, we can observe that there is no queries being reused, i.e., there are no numbers in the bold italic font. This corresponds to the five queries for which our algorithm decides not to reuse any previous query plans.
Table 6.2: Query Results for Query Plans from PDC

<table>
<thead>
<tr>
<th>Group1</th>
<th>Correct</th>
<th>Complete</th>
<th>Group2</th>
<th>Correct</th>
<th>Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seed A</td>
<td>N/A</td>
<td>N/A</td>
<td>Seed B</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Query</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>100%</td>
<td>95%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>100%</td>
<td>69%</td>
<td>2.1</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>1.3</td>
<td>100%</td>
<td>95%</td>
<td>2.2</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>1.4</td>
<td>100%</td>
<td>100%</td>
<td>2.3</td>
<td>100%</td>
<td>97.7%</td>
</tr>
<tr>
<td>1.5</td>
<td>100%</td>
<td>80%</td>
<td>2.4</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Group3</td>
<td>Correct</td>
<td>Complete</td>
<td>Group4</td>
<td>Correct</td>
<td>Complete</td>
</tr>
<tr>
<td>Seed C</td>
<td>N/A</td>
<td>N/A</td>
<td>Seed D</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Query</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>100%</td>
<td>100%</td>
<td>4.1</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>3.2</td>
<td>100%</td>
<td>100%</td>
<td>4.2</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>3.3</td>
<td>100%</td>
<td>100%</td>
<td>4.3</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>3.4</td>
<td>100%</td>
<td>100%</td>
<td>4.4</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>3.5</td>
<td>100%</td>
<td>100%</td>
<td>4.5</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>
**Query Planning Overhead:** The QPCache query planning algorithm incurs some overhead compared with the normal bidirectional algorithm. But this overhead is very small (about few milliseconds), and is negligible compared with the time saved from query execution, which is usually several seconds.

**Comparing Actual Query Results:** From the comparison of query execution time and query plan score we observe that, most of the time, the query plans generated by PDC run faster than the query plans generated by DDC. At the same time, the score of the plans from PDC is *not* significantly lower than the score of the plans from the original planning algorithm. Since a very important metric of query plan is the actual results obtained from the execution of the plan, now we compare the query results we obtained using the query plan generated by PDC with the results generated by DDC. The goal here is to examine if the results from PCD (using our reusing strategy) are *correct* and *complete*.

Table 6.2 shows the results. First, we can observe that for all the queries we consider, the query results obtained from the plan generated by our reusing strategy PCD are always 100% *correct*. Second, except for 5 queries (in bold font), the results from PDC are also 100% *complete*. Among the five queries with incomplete answers, 3 of them have nearly 100% complete answers (above 95%). After careful examination of the query plans from PDC, we found that the incompleteness mainly comes from three search terms. These are *Organism*, *SNPID*, and *Gene Ontology*. For the 5 queries with incomplete answers, the query plans generated by PDC reuse some data sources that have a low coverage data on the above three search terms. For example, given a gene name, say **ERCC6**, the *Gene* data source returns up to 8 organisms from which this gene could come, but *dbSNP* data source only gives 4. This turns out to be the case for queries 1.1 and 1.3. Similarly, the *Seattle* data source returns much smaller number of SNPs located in the gene than *dbSNP* (query 1.2 and 1.5); and the
SNP500 data source returns less number of Gene Ontology terms than the Gene data source (query 2.3).

Overall, the above statistics show that for most cases, the query plans generated by PCD obtain correct and complete answers and at the same time have much shorter execution time.

6.6 Summary

In this chapter, we have presented a query-plan-driven query caching strategy for accelerating searches over deep web data sources. Our strategy involves not only caching previously extracted data, but more importantly, caching and utilizing the previously executed query plans. The QPCache algorithm we have developed generates a detour query plan for a new query, which reuses cached query plans, and thus increases the possibility of reusing cached data.

Our experiments show that our query caching strategy is able to accelerate the execution of keyword search queries significantly, as compared with a pure data caching schema or a scenario where no caching is done. Furthermore, our algorithm can detect situations where reusing query plans results in low-quality plans, and thus achieve the right trade-off between plan quality score and execution time speedup.
CHAPTER 7
SUPPORT FAULT TOLERANCE FOR DEEP WEB SEARCHERS: AN INCREMENTAL QUERY PLAN GENERATION APPROACH

Data access over wide-area networks can involve a large number of remote data sources and communication links, both of which are subject to congestion and failures. This can cause unpredictable unavailability and/or inaccessibility of sources [124]. In turn, it can lead to disruption in query processing capabilities of the integrated system. Clearly, for providing better experience and accessibility to information to the users, it is very important for a deep web integration system to gracefully handle situations where some data sources may become unavailable and/or inaccessible. In this chapter, we propose a solution to maintain query processing capability of an integrated deep web system in the presence of unavailable or inaccessible data sources. Our solution involves dynamically adapting query processing when unexpected data source unavailability or inaccessibility is detected. We exploit the data redundancy that is found across deep web data sources, i.e., many deep web data sources in a particular domain could overlap in terms of data content. We incrementally generate a partial new query plan by bringing in new data sources that were not in the original query plan to replace the subplan that became inaccessible.
7.1 Motivation

The solution for supporting fault tolerance proposed in this chapter is driven by the following observation. Deep web data sources often have data redundancy, i.e., the same data can be obtained from multiple data sources [15]. For example, in the biological domain, the gene information about a gene can be obtained from multiple deep web data sources such as NCBI Gene, Human Protein, and HOGO Gene. In commercial domains also, identical information often can be retrieved from many data sources. The data redundancy we consider is not limited to just complete duplication of information, such as when there are mirrored data sources. But, we also consider what can be referred to as partial redundancy, i.e., when two data sources may overlap only in terms of certain attributes. Overall, the premise of our approach is that data redundancy across deep web data sources can be used to hide unavailable data sources from the users of an integrated system, thus improving access to information and providing better user experience.

The following example further explains how data redundancy can be used to address the challenges introduced by unavailable data sources.

**Example:** A biologist interested in Genetic studies issues a query Q1={Chromosome10, SNP_Frequency, MolA, Protein_Interaction}. This query aims to obtain the MolA information and the protein interaction of the genes that are located at Chromosome 10, and the SNP frequency data of the SNPs (Single Nucleotide Polymorphisms) that are located at Chromosome 10. Let us assume for now that such a query can be expressed as a keyword query, and the system can correctly interpret the semantics of this query. A query plan which could answer Q1 is shown as sub-figure (a) of Figure 7.1. Here, each node represents a deep web data source and each link represents the dependence between data sources. The term above each link shows the attribute obtained from the source of the edge, and used for querying the sink of the edge. To
answer the query $Q_1$, SNP500 data source takes the chromosome number as input and returns the SNPs and genes located in this chromosome. Seattle data source takes the SNPs from SNP500 as input and returns the requested SNP frequency information and the gene IDs in which the SNPs are located. Similarly, BOND data source takes the gene name from SNP500 as input and returns the MolA value. Finally, Human Protein data source takes the gene IDs from Seattle as input and returns the protein interaction.

Now, suppose that the data source Seattle is temporarily unavailable due to hardware problems. Since the data source BOND is independent of the unavailable data source Seattle, the unshaded part of the plan in sub-figure(a) will be processed normally. The query processing, however, cannot proceed from Seattle and we cannot obtain the SNP frequency and gene ID information from Seattle. We further assume that Seattle is the only data source that can provide gene ID among the data sources integrated in the system. As a result, the unavailability of Seattle makes the data source Human Protein unusable as well. Here, we want to emphasize that Seattle is the only unavailable data source. Human Protein is available, but is unusable because the unavailability of Seattle makes it impossible for us to obtain the necessary input of Human Protein.

In this case, the current query plan can only answer part of $Q_1$, which is given a chromosome number, we can obtain the MolA information (from BOND) for the genes located in the chromosome and the SNPs (from SNP500) located in the chromosome. The missing part of this query plan (the information we are supposed to obtain from the two unusable data sources) is that given a SNP, how to obtain its frequency and protein interaction information. In order to recover this missing part, we can generate a new partial keyword query $Q_2=\{\text{rs7412, Frequency, Protein\_Interaction}\}$ (rs7412 is a SNP located in Chromosome 10 which can be obtained from the results of SNP500)
and generate a partial new query plan to replace the unusable part of the original plan.

The incremental query plan we generate is shown as sub-figure (b). In this new plan, we utilize the data redundancy between data source Seattle and dbSNP, as well as the data redundancy between the data sources Human Protein and NCBI Gene. In the first case, both of them take SNPs as input and return SNP frequency information, whereas in the second case, both of them return the protein interaction information.

Overall, in this chapter, we are proposing an approach which can be considered as data redundancy based incremental query processing. During the processing of the original query plan, if some data sources become unavailable, the simplest action is to completely discard the current execution, and generate a new plan for the original query. The obvious disadvantage of this approach is that it wastes a significant amount of work has already been performed. In our approach, the query processing
will not be terminated. Instead, the execution of the part of the query plan which becomes unusable due to the unavailable data sources will be suspended, and we will incrementally generate a partial new query plan for the unusable part in the original query plan based on data redundancy across data sources.

In literature, systems and algorithms have been developed to address the problems of query processing in a dynamic network environment. Query scrambling [124, 3] reacts to unexpected delays by rescheduling the operations of a query during its execution and hides delays encountered. Proactive alternative query plan generation builds alternative query plans that may be needed due to runtime variations in the environment during compile-time [105]. Existing approaches for handling unpredictable delays are based on re-ordering of existing operators or creating new operators without introducing new data sources into the query plan. In other words, the basic assumption in existing approaches is that delays are tolerable. However, in our approach, unlike the existing work, we are able to take advantage of data redundancy, and can handle complete unavailability of a data source at the runtime.

7.2 Data Redundancy Model

In this section, we introduce the data redundancy model we consider in this chapter. Throughout this chapter, we use the same data source model which is introduced in Section 7.2 and the formulation for queries and query plan as described in Section 6.2.1.

We say that there is data redundancy between two data sources if their usage can be overlapping. The usage of a data source is captured by the data source model we introduced above. For example, in the motivating example in Section 1, the usage of the data source SNP500 is that given a chromosome number (input), we can obtain SNP IDs and gene names (outputs) related with this chromosome.
Formally, the data redundancy between a set of deep web data sources is captured through a redundancy graph, $RG = (V, E)$. Here, $V$ represents the set of all data sources in the system and $E$ is the set of redundancy edges between these data sources. A redundancy edge $e_{AB}$ between the data sources $A$ and $B$ shows that there is data redundancy between $A$ and $B$. Such an edge is introduced if the following three conditions hold: 1) The input attribute set ($IN$ in the data source model) of $A$ and $B$ are the same, 2) there are common output attributes between $A$ and $B$ ($O_A \cap O_B \neq \emptyset$); and 3) the conditions specified by the $C$ columns in the data source model of $A$ and $B$ do not contradict each other. The reason we need the third condition can be explained with the following example. If $A$ provide human gene information and $B$ provides mouse gene information, even if $A$ and $B$ may share the same input and output attributes, there is no data redundancy.

In the graph representation, an edge $e_{AB}$ is annotated by the overlapping input and output attributes between $A$ and $B$. Formally, $IN_e = \{c|c \in (IN_A \cap IN_B)\}$ and $O_e = \{c|c \in (O_A \cap O_B)\}$. Figure 7.2 shows the redundancy graph model among four data sources.
7.3 Problem Formulation

This section gives a formal description of the problem we are focusing on.

7.3.1 Incremental Query Plan Generation Problem

We are considering the problem associated with one or more unavailable and/or inaccessible data sources. To formulate the problem, we use the following definitions.

**Data Source Dependency:** In a query plan $P$, we consider a data source $B$ to be dependent on the data source $A$ if and only if there is a path in $P$ from $A$ to $B$.

**Query Subplan:** A query subplan $SubP$ of an original query plan $P$ is a sub-graph of the original query plan graph. Formally, $SubP = (V', E', V_0')$ where $V' \subseteq V$, $E' \subseteq E$, $V_0' \subseteq V$ and $|V_0'| > 0$.

**SubQuery:** Given a query $Q$ which contains $n$ search terms and a query subplan $SubP$ of the query plan of $Q$, the subquery corresponding to the subplan $SubP$ is a query containing a set of entity name terms $SubET$ and a set of attribute terms $SubAT$, computed with the following constraints: 1) $SubET$ and $SubAT$ are subsets of $Q$, 2) $SubET$ can only contain entity names that correspond to the input attributes of the starting data sources in $SubP$, and 2) $SubAT$ can only contain attributes covered by the output attribute sets of the data sources in $SubP$.

For example, the query $Q2=\{rs7412, Frequency, Protein Interaction\}$ in the motivating example in Section 1 is the subquery corresponding to the shaded subplan (Seattle and Human Protein) in the sub-figure (a) of the Figure 7.1. The input attribute of the starting data source (Seattle) in the subplan is SNPID, so rs7412, which is an entity name of an SNP, is the entity name term in the subquery $Q2$. The attribute terms Frequency and Protein Interaction in $Q2$ are requested in the original query $Q1$ and are also covered by the outputs of the two shaded data sources in the subplan.
Impacted Subplan: Given a query plan $P$ that is generated for a given query, and let $UDS$ be the set of unavailable data sources, as detected during the execution of $P$. We define a subplan of $P$, $ISubP$, to be the impacted subplan if 1) the set of starting data sources of $ISubP$ equals the unavailable data source set, $UDS$, and, 2) every data source node in $ISubP$ is dependent on at least one data source in the set $UDS$.

In other words, the impacted subplan contains all data sources which could be reached from one or more of the unavailable data sources.

As an example, for the query plan shown in Figure 7.3, suppose the data source $B$ is unavailable. The subplan enclosed by a rectangle with dotted border in the upper part of the figure (plan with nodes $B$, $D$, $E$ and $F$) is the impacted subplan of $B$.

Intuitively, we should consider replacing the impacted subplan with a new plan to fix the original query plan. However, in view of the existing data redundancies, the impacted subplan may not always need to be replaced. Consider the following scenario. Suppose under the given data redundancy graph $RG$, the data source $H$ has the same usage as the data source $B$, i.e., $H$ also takes the term $t1$ as input and covers the term $t3$ and $t4$ in its output. Then, we do not need to replace the entire impacted subplan. Instead, we only need to replace the data source $B$ with $H$, and the remaining data sources in the impacted subplan can still be used.

More generally, from the example above, the reason we can keep other data sources in the impacted subplan is that these data sources are not necessarily unusable after failure of $B$, in view of the existing redundances. To capture this, we formally introduce the notion of Minimal Impacted Subplan.

Minimal Impacted Subplan: Given a data redundancy graph $RG$, a subplan $MISubP$ is the minimal impacted subplan of the plan $P$ if the following two conditions hold: 1) $MISubP$ is a subplan of the impacted subplan of $P$; 2) There is no
data source \(D\) in \(MISubP\) that is still \textit{usable}, considering data redundancies captured through \(RG\). For example, the subplan with a single data source \(B\) in Figure 7.3 is the \textit{minimal impacted subplan} under the given redundancy graph \(RG\).

Given a \textit{minimal impacted subplan} \(MISubP\), we can obtain the subquery corresponding to \(MISubP\). Then, our goal is to generate another query plan to answer the subquery, and have this plan replace \(MISubP\). But, it is likely that for some user requested attributes, all data sources that can provide them are either unavailable or unusable. This means that we definitely cannot obtain the values of these attributes. We define these attributes as the \textit{dead attributes}. In view of this, our goal becomes to generate a new query plan for the \textit{Maximal Fixable Subquery} of \(MISubP\), which is defined as follows.

**Maximal Fixable Subquery:** Given a minimal impacted subplan, \(MISubP\), the Maximal Fixable Subquery, \(MaxSubQuery\), contains a subset of the search terms in the subquery corresponding to \(MISubP\), such that any search term that is not in \(MaxSubQuery\) but in the subquery corresponding to \(MISubP\), is a \textit{dead attribute}.

### 7.3.2 Problem Description

Using the above definitions, our incremental query plan generation problem is formally stated as follows. For a query \(Q\), an original query plan \(P\) is generated. During the execution of plan \(P\), at some time, a set of data sources \(UDS\), which is defined to be the \textit{unavailable data source set}, in \(P\) becomes unavailable. Based on a data redundancy graph \(RG\), we want to find a \textit{minimal impacted subplan}, denoted as \(MISubP\). Then, we want to take advantage of the data redundancy between the data sources in \(MISubP\) and other deep web data sources not in the original query plan \(P\) to incrementally generate a new partial query plan to replace \(MISubP\), such that that the resulting new plan could answer the \textit{maximal fixable subquery} of \(MISubP\).
To solve the above problem, we take the following steps.

1. Given a query plan $P$, a data redundancy graph $RG$, and the unavailable data source set $UDS$, we need to find the minimal impacted subplan $MISubP$.

2. Given a $MISubP$, we need to generate the maximal fixable subquery of $MISubP$.

3. We need to generate a query plan for the maximal fixable subquery.

The next section describes these steps in details.

### 7.4 Incremental Query Planning Algorithm

In this section, we first describe the algorithm which finds the *minimal impacted subplan* from a query plan when there are unavailable data sources. Then, we describe how a *maximal fixable query* is generated from a *minimal impacted subplan*. Finally, we show how an *incremental partial query plan* is generated.
7.4.1 Minimal Impacted Subplan Generation

In this section, we will first give an overview of the minimal impacted subplan generation algorithm. Then, we describe the details of the algorithm.

**Algorithm Overview**

The first step is to find the *impacted subplan*, which can be done as follows. Given a query plan and a set of unavailable data sources, $UDS$, we perform a simple traversal along the edges in the query plan starting from the data sources in $UDS$. Consider the example shown in Figure 7.3, and suppose $UDS = \{B, I\}$. We can see that the impacted subplan is a graph with two *disconnected components*, both of which are enclosed by dotted bordered rectangles. The first component contains $B$, $D$, $E$, and $F$, and the second component contains $I$, $J$, $K$, and $L$. The above impacted subplan is shown as sub-figure (a) in the Figure 7.4. The terms above each edge in this figure are the input attributes of the data source pointed by the edge. The terms inside each data source node are the terms which can be obtained from the data source.

The set of input attributes of a data source $D$ is denoted by $IA_D$. For example, the input of data source $B$ is the term $t_1$ ($IA_B = \{t_1\}$) and we can obtain the value of attribute $t_3$ and $t_4$ by querying $B$.

Now, we consider how we can obtain the *minimal impacted subplan*. According to the definition, we need to determine the *usable* data sources from the impacted subplan graph, and delete them to obtain the *minimal impacted subplan*. In order to find the *usable* data sources in an impacted subplan, we need to begin from the data sources in $UDS$. In our example, $B$ is in $UDS$, so $B$ is definitely *unusable*. Then we need to decide if $B$ being unusable will make the intermediate descendants of $B$ ($D$ and $E$) unusable as well. If there exists a data source, say $H$, that has data redundancy with $B$, specifically, $H$ takes the term $t_1$ as input and has the term $t_3$
as output, we know that $D$ will still be usable. If there is no data source taking the term $t1$ as input and has the term $t4$ as output, we know that $E$ is unusable due to the un-usability of $B$.

We define the data sources in the set $UDS$ to be self-crashed data sources, and the data sources that have become unusable due to the un-usability of their ancestors to be cascading-crashed data sources. In the above example, data sources $B$ and $I$ are self-crashed data sources, and $E$ is a cascading-crashed data source. From the definition of minimal impacted subplan in Section 7.3, we know that a minimal impacted subplan should not contain any usable data source, i.e. it has to be composed only of the unusable data sources. Thus, the minimal impacted subplan is composed of the self-crashed data sources and the cascading-crashed data sources in the corresponding impacted subplan.

**Redundancy Checking Algorithm**

From the discussion above, we know that, to find the minimal impacted subplan, we need to identify the self-crashed and cascading-crashed data sources from the impacted subplan. Self-crashed data sources are easy to identify because they are the data sources in the unavailable data source set $UDS$. To efficiently find the cascading-crashed data sources in an impacted subplan, we proposed a redundancy checking algorithm.

Given an impacted subplan, the redundancy checking algorithm traverses the impacted subplan graph and finds a subgraph which only contains self-crashed and cascading-crashed data sources. This subgraph is the minimal impacted subplan. The traversal begins from the self-crashed data source (data sources in the set $UDS$) in
the impacted subplan graph. The redundancy checking method is shown as Algorithm 7.4.1. The input parameters include \( RG \), which is the redundancy graph, and \( UDS \), which is the set of self-crashed data sources.

Initially, we build a \textit{crashed nodes} queue to store all the \textit{crashed} nodes. Initially, the \textit{self-crashed} nodes are visited by the redundancy checking algorithm and put into the crashed nodes queue. For each node in the crashed nodes queue, we perform a \textit{redundancy check}, which we will elaborate later. Suppose we are currently examining a crashed node \( D \) and we want to know whether the crashing of \( D \) would lead to cascading-crashes of the data sources dependent on \( D \). For this purpose, all the outgoing neighbors of \( D \) in the impacted subplan are checked by the algorithm. If the algorithm finds that none of \( D \)'s dependent data source is \textit{cascading-crashed}, the graph traversal stops. Otherwise, if any of the dependent data source of \( D \), say the data source \( X \), is \textit{cascading-crashed}, we add the data source \( X \) to the \textit{crashed nodes queue} and mark \( X \) as \textit{crashed}. 
Algorithm 7.4.1: Redundancy-Checking\textit{(RG, UDS)}

initialize crashed nodes queue CNQ
add data sources in UDS to CNQ

\textbf{while} CNQ\ne\emptyset

\hspace{1em} D = CNQ.pop()

\hspace{1em} mark \hspace{1em} D as crashed

\hspace{1em} /*graph traversal, examine dependent nodes of D*/

\hspace{1em} \textbf{foreach} \hspace{1em} X \in D.dependentdatasources

\hspace{2em} same\_usage\_node = false

\hspace{2em} /*iterate over neighbors in the redundancy graph*/

\hspace{2em} \textbf{foreach} \hspace{1em} H \in \text{Neighbors}(RG, D)

\hspace{3em} denote the edge between H and D in the redundancy graph as \( e_{DH} \)

\hspace{3em} /*the first two conditions*/

\hspace{3em} \textbf{if} \hspace{1em} H \text{ is not crashed}

\hspace{4em} /*the third condition*/

\hspace{5em} \textbf{if} \hspace{1em} IN_H = IN(e_{DH}) \hspace{1em} \textbf{and} \hspace{1em} IN_X = O(e_{DH})

\hspace{5em} same\_usage\_node = true

\hspace{5em} \textbf{break}

\hspace{5em} \textbf{if} \hspace{1em} same\_usage\_node = false

\hspace{6em} CNQ.push(X) /*X is cascading crashed*/

The key component of the algorithm is the redundancy checking procedure. Given a pair of data sources \( D \) and \( X \), where \( X \) is dependent on \( D \) and \( D \) is \textit{crashed}, the redundancy checking procedure will conclude the dependent data source \( X \) is \textit{not cascading-crashed} if all of the following conditions hold: 1) In the \textit{redundancy graph}, data source \( D \) has neighbors; 2) Among all the such neighbors, there is at least one neighbor has not been identified as \textit{crashed} (either \textit{self-crashed} or \textit{cascading-crashed}),
and 3) Among all the such neighbors that are not crashed, at least one data source has the *same usage* as the data source $D$. The first two conditions ensure that there are *non-crashed* data sources that share redundancy with the crashed data source $D$.

We say that a data source $H$ has the *same usage* as data source $D$, if the edge $e_{DH}$ between data source $D$ and $H$ in the redundancy graph has input annotation equal to $IN_D$, the input attributes of crashed data source $D$, and the output annotation equal to $IN_X$ (the input attributes of $D$’s dependent data source). In other words, the data source $H$ takes the same input attributes as $D$, and provides the input attributes of $D$’s dependent data sources.

When all the crashed nodes in the crashed nodes queue have been examined by the graph traversal and redundancy checking procedures described above, the algorithm terminates. All visited data sources and the edges connecting them form the minimal impacted subplan.
Example: Figure 7.4 shows an example of the execution of the redundancy checking algorithm described above. In Figure 7.4, sub-figure (a) shows the impacted subplan graph, sub-figure (b) shows a part of the redundancy graph (components related to data sources I, B and E), and sub-figure (c) shows the minimal impacted subplan generated by our algorithm. The redundancy checking algorithm starts to traverse the two disconnected components of the impacted subplan graph from the two self-crashed data sources, B and I. Let us consider the traversal from the data source B first. B has two dependent data sources, D and E. Consider the step where the algorithm tries to determine whether the data source D is cascading-crashed due to the crash of B. For this, the algorithm first finds the data sources having data redundancy with B from the redundancy graph, which are H, P, and Q. None of them are crashed. Next, the algorithm checks which data sources among H, P and Q have the exact usage as the data source B, with respect to D in the impacted subplan. We notice that the edge $e_{BH}$ in the redundancy graph has the input annotation $t_1$ and output annotation $t_3$, which match the input of the data source B and the input of data source D, respectively. As a result, H has the exact same usage as B, with respect to D. Therefore, D is not cascading-crashed due to the crash of B. According to the algorithm, the traversal to data source D is terminated.

Then, the algorithm examines the other dependent data source of B, which is E. If we want to show data source E is not cascading-crashed, we need to find a data source which shares redundancy with B and the redundancy edge has input annotation of $t_1$ (the input attribute of B) and output annotation of $t_4$ (the input attribute of E). We notice that among the three data sources sharing redundancy with B (H, P, and Q), none of them has the exact same usage as B with respect to E. As a result, the data source E is cascading-crashed and E is added to the crashed nodes queue. Then, the algorithm continues to traverse from the data source E. In a
similar fashion, $F$ is determined cascading-crashed due to the crash of $E$. With this, the minimal impacted subplan originating from $B$ has been found. The redundancy checking algorithm also traverses the second component in the impacted subplan, which has the data source $I$ as the root. In the redundancy graph, data source $P$ shares redundancy with $I$, and $P$ has the exact same usage as the data source $I$, with respect to the data source $J$. As a result, the data source $J$ is not cascading-crashed and the traversal to $J$ is terminated. The final generated minimal impacted subplan is shown in sub-figure (c).

### 7.4.2 Maximal Fixable SubQuery Generation

In Section 7.4.1, we mentioned that a minimal impacted subplan may contain two types of data sources, which are the self-crashed data sources and the cascading-crashed data sources. In this section, based on the types of data sources a minimal impacted subplan contains, we categorize the fixability of a minimal impacted subplan into four types. Then for each type of minimal impacted subplan, we present a method to generate the maximal fixable subquery. Here, we want to emphasize that if there are disconnected components in a minimal impacted subplan, we consider each component as a separate minimal impacted subplan of the original query plan. For example, the minimal impacted subplan in the sub-figure (c) in Figure 7.4 has two disconnected components. Thus, we will generate two maximal fixable subqueries, i.e. one for each of the components. Further, we will also need two new partial query plans.

**Fixability Categorization of Minimal Impacted Subplan**

When the minimal impacted subplan of a query plan with unavailable data sources is identified, our next goal is to generate a new partial query plan to replace the minimal
impacted subplan. The method by which a new partial query plan is generated depends on the fixability of the minimal impacted subplan, i.e., how much the minimal impacted subplan can be fixed using other data sources. We categorize the fixability of a minimal impacted subplan into four types that we will elaborate below.

Prior to that, recall that in Section 7.3, we define an attribute to be a dead attribute if the following conditions hold: 1) the attribute is requested by the user query; 2) the data source providing the value of this attribute in the original query plan is crashed and 3) from the redundancy graph, we cannot find another data source, which is not crashed, and could cover this attribute as its output. This definition shows that a dead attribute definitely cannot be recovered.

**Fully-Fixable:** Consider an original query plan $P$ and a minimal impacted subplan $MISubP$. If $MISubP$ only contains self-crashed data sources, and furthermore, none of the output attributes requested by the user query and covered by $MISubP$ is a dead attribute, we consider $MISubP$ to be a fully-fixable minimal impacted subplan.

**Partial-Fixable:** Consider an original query plan $P$ and a minimal impacted subplan $MISubP$. If $MISubP$ only contains self-crashed data sources, but there is at least one output attribute requested by the user query and covered by $MISubP$ is a dead attribute, we consider $MISubP$ to be a partial-fixable minimal impacted subplan.

**Cascading-Fully-Fixable:** Consider an original query plan $P$ and a minimal impacted subplan $MISubP$. If $MISubP$ contains cascading-crashed data sources, and furthermore, none of the output attributes requested by the user query and covered by $MISubP$ is a dead attribute, we consider $MISubP$ to be a cascading-fully-fixable minimal impacted subplan.

**Cascading-Partially-Fixable:** Consider an original query plan $P$ and a minimal impacted subplan $MISubP$. If $MISubP$ contains cascading-crashed data sources, and there is at least one output attribute requested by the user query and covered by
MISubP is a dead attribute, we consider MISubP to be a cascading-partial-fixable minimal impacted subplan.

Finding Maximal Fixable Subquery

Algorithm 7.4.2: MFSubquery

/*MISubP is the input minimal impacted subplan*/
/*MFQ is the result maximal fixable subquery*/
add self-crashed data sources (without incoming edges) to a queue Q
while Q≠Φ
    D = CNQ.pop()
    find IN(D), RO(D), L(D)
    if D.incomingedge()=Φ
        MFQ.ET.add(IN(D)) /*entity terms of MFQ*/
    if MISubP with partial fixability
        foreach t ∈ RO(D)
            if t is not a dead attribute
                MFQ.AT.add(t) /*attribute terms of MFQ*/
                /*we only consider terms which are recoverable*/
            if D.outgoingedge()=Φ
                MFQ.AT.add(L(D)) /*attribute terms of MFQ*/

Given a minimal impacted subplan, we want to find the maximal fixable subquery, for which we will generate a new partial query plan. Similar to the original query defined in Section 6.2.1, a maximal fixable subquery contains a set of search terms that could be partitioned into two sets, the entity term set ET and the attribute term set AT. In this section, we introduce an algorithm, MFSubquery algorithm.
<table>
<thead>
<tr>
<th>Fixability Categorization</th>
<th>Maximal Fixable Subquery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully-Fixable</td>
<td>$MFQ = {(ET, AT)</td>
</tr>
<tr>
<td>Partially-Fixable</td>
<td>$MFQ = {(ET, AT)</td>
</tr>
<tr>
<td>Cascading-Fully-Fixable</td>
<td>$MFQ = {(ET, AT)</td>
</tr>
<tr>
<td>Cascading-Partial-Fixable</td>
<td>$MFQ = {(ET, AT)</td>
</tr>
</tbody>
</table>
(Algorithm 7.4.2), to construct the ET and AT sets, considering minimal impacted plans with different types of fixability.

This algorithm traverses all data source nodes in a minimal impacted subplan in a breath-first manner starting from the self-crashed nodes. When a data source $D_i$ is visited, the algorithm builds three sets for $D_i$:

- The input set of $D_i$, denoted as $IN(D_i)$, which contains the input attributes of $D_i$,

- The requested output set of $D_i$, denoted as $RO(D_i)$, which contains the output attributes of $D_i$ that are also attribute terms in the original user query, and

- The linking set of $D_i$, denoted as $L(D_i)$, which contains the output attributes of $D_i$ that are used as input of $D_i$'s dependent data sources.

For example, for data source $B$ in the subplan shown as sub-figure (c) in Figure 7.4, we have $IN(B) = \{t1\}$, $RO(B) = \{t11\}$ and $L(B) = \{t4\}$.

For the maximal fixable subquery, the ET set should contain the entity name terms that are used to initiate the query answering. If we consider the minimal impacted subplan as a black box, the input of the black box is the input set ($IN$ set) of all data sources in the minimal impacted subplan that do not have any incoming edges. Note that this is also the set of self-crashed data sources. As a result, during the traversal of the MFSubquery algorithm, all terms in the $IN$ set of self-crashed data sources should be added to the set $ET$.

The AT set contains the attribute terms whose values should be obtained by the minimal impacted subplan. Again, if we consider the minimal impacted subplan as a black box, we need to obtain values of terms of two types. The first type contains the attributes that are explicitly requested in the user query. This is also the attributes in the $RO$ set of all data sources in the minimal impacted subplan. The second type
contains the attributes that link the minimal impacted subplan to the data sources that are dependent on them. This is also the linking set \( L \) of all the data sources in the minimal impacted subplan which have outgoing edges to the data source nodes outside the minimal impacted subplan. One complicating issue, however, arises for the two types of partial-fixable plans. We know some attributes in the \( RO \) set cannot be obtained from other data sources, i.e., they are dead attributes. As a result, when constructing the set \( AT \), we need to exclude such dead attributes.

In summary, during the traversal of the MFSubquery algorithm, we add all terms in the \( RO \) set of all data sources (for plans with partial fixability, excluding the dead attributes) and all terms in the \( L \) set of data sources to the \( AT \) set.

For example, for the plan involving data sources \( B, E \) and \( F \) in sub-figure (d) in Figure 7.4, suppose this is a cascading-fully fixable plan. The maximal fixable subquery would be \{\( t_1, t_{11}, t_{14} \)\}, where \( ET = \{t_1\} \) and \( AT = \{t_{11}, t_{14}\} \). Terms \( t_4 \) and \( t_6 \) are not in the maximal fixable subquery because they are not in the linking sets of data sources having outgoing edges to the data sources outside of the minimal impacted subplan. Similarly, for the plan involving data sources \( I \) in the sub-figure (c) in Figure 7.4, suppose this is a partial fixable plan (\( t_{12} \) is a dead attribute), the maximal fixable subquery would be \{\( t_7, t_8 \)\}, where \( ET = \{t_7\} \) and \( AT = \{t_8\} \). \( t_{12} \) is excluded from the subquery because we definitely cannot obtain this term from other data sources. Since \( t_8 \) is in the linking set of \( I \), which has an edge link to the data source \( J \). \( J \), in turn, is outside of the minimal impacted subplan. Thus, \( t_8 \) is in the \( AT \) set.

We summarize the maximal fixable queries for the four categories of fixability in Table 7.2.
7.4.3 Incremental Query Plan Generation

In Chapter 5, we proposed a bidirectional query planning algorithm for generating query plans for cross-source queries over deep web data sources. In this approach, a dependency graph capturing the inter-dependence among deep web data sources is built and the query planning problem is considered to be a problem of finding the minimal subgraph of the dependency graph which covers all the query terms. We show that the query planning problem is NP-hard and we proposed a bidirectional graph traversal algorithm with several heuristics to find a query plan. In our system, given a maximal fixable subquery $MFQ$, we consider it as an ordinary user query and we adapt the previously proposed bidirectional query planning algorithm to generate a query plan for $MFQ$ with following modification: we exclude the data sources which are in the minimal impacted subplan from the dependency graph, as a result, we could generate a query plan for the $MFQ$ without using any crashed data sources.

7.5 Evaluation

In this section, we describe the experiments we conducted to evaluate our techniques. Our evaluation shows that our incremental query planning generation approach not only speedups the process of dealing with inaccessible data sources, but also that the incremental query plans generated by our approach achieves high quality results. Furthermore, we show that the incremental query plan generation mechanism introduced in Section 7.4 only has a very small overhead.

7.5.1 Experiment Setup

Our evaluation was done using 12 biological deep web databases we have integrated, which includes dbSNP, Entrez Gene, Protein, BLAST, SNP500, Seattle, SIFT, BIND,
Human Protein, HGNC, Mouse SNP, and ALFRED. The input and output schema of the data sources are extracted using a previously created wrapper. Our collaborating biologist provides us 20 queries for our evaluation. The 20 queries were divided into 4 groups, with each group having 5 queries that represent one type of fixability that we had defined in Section 7.4.2. Recall that the four types are Fully-Fixable (FF), Partial-Fixable (PF), Cascading-Fully-Fixable (CFF), and Cascading-Partial-Fixable (CPF).

For each user query, an original query plan is generated using the bidirectional query planning algorithm proposed in Chapter 5. Then, the original query plan is executed. During the query processing, we randomly select data source(s) in the plan to be unavailable. This simulates unavailability in the real environment. Then, the methodology proposed in this paper is used to deal with the unavailability of the crashed data sources.

For the purpose of comparison, we also use a baseline method. The baseline method deals with crashed query plan as follows. During plan execution, When some data source(s) crashed, the baseline method stops the execution of the plan, and generate a new query plan from scratch. The difference in the two approaches is that this baseline method may not at all use the information that had already been retrieved before the crash.

We conducted three experiments to evaluate our approach. In the first experiment, we record the query answering time for each experiment query using the two fault tolerance methods: the baseline method and our incremental plan generation method. The query answering time is the duration starting from the user issuing a query to the time the answer of the query is returned to the user. In the second experiment, we show the actual query results obtained by the query plan generated incrementally and
evaluate the quality of the results. In the last experiment, we examine the overhead of our incremental approach.

### 7.5.2 Experimental Results

![Comparison of Query Answering Time](image)

Figure 7.5: Comparison of Query Answering Time: (a) Answering Time Comparison for FF Case, (b) Answering Time Comparison for PF Case, (c) Answering Time Comparison for CFF Case, (d) Answering Time Comparison for CPF Case.
Table 7.3: Incremental Query Plan Result Quality Analysis

<table>
<thead>
<tr>
<th>Type</th>
<th>Ideal Recall</th>
<th>Real Recall</th>
<th>Type</th>
<th>Ideal Recall</th>
<th>Real Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF1</td>
<td>100%</td>
<td>100%</td>
<td>CFF1</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>FF2</td>
<td>100%</td>
<td>100%</td>
<td>CFF2</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>FF3</td>
<td>100%</td>
<td>100%</td>
<td>CFF3</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>FF4</td>
<td>100%</td>
<td>100%</td>
<td>CFF4</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>FF5</td>
<td><strong>100%</strong></td>
<td><strong>52%</strong></td>
<td>CFF5</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>PF1</td>
<td>95.8%</td>
<td>95.8%</td>
<td>CPF1</td>
<td>86%</td>
<td>86%</td>
</tr>
<tr>
<td>PF2</td>
<td>97%</td>
<td>97%</td>
<td>CPF2</td>
<td>96%</td>
<td>96%</td>
</tr>
<tr>
<td>PF3</td>
<td>67%</td>
<td>67%</td>
<td>CPF3</td>
<td>97.6%</td>
<td>97.6%</td>
</tr>
<tr>
<td>PF4</td>
<td>75%</td>
<td>75%</td>
<td>CPF4</td>
<td>97.6%</td>
<td>97.6%</td>
</tr>
<tr>
<td>PF5</td>
<td><strong>99%</strong></td>
<td><strong>55%</strong></td>
<td>CPF5</td>
<td>87.5%</td>
<td>87.5%</td>
</tr>
</tbody>
</table>

**Comparing Query Answering Time:** Figure 7.5 shows the comparison of the query answering time between the two methods (baseline and incremental), consider the queries from each of the four categorizations of fixability. The x-axis in each sub-figure is the query ID. For the y-axis, the query answering time for the baseline method is always considered as the reference, i.e., as the value of 1. The answering time for the incremental method is the ratio between the answering time of the incremental method and the answering time of the baseline method. We have the following observations. First, except for the third query in the FF case, for all other queries in all fixability categorizations, the query answering time for the incremental method is smaller than the query answering time for the baseline method. For the third query in the FF case, because our adaptive method uses a data source with long response time (but one that is not considered as inaccessible), the query answering
time of our adaptive method is longer than the baseline method. Second, for 12 out of 20 queries, our approach gives more than 20% speedup, and for 9 out of 20 queries, our approach gives more than 40% speedup.

**Comparing Actual Query Result:** Table 7.3 shows the actual results obtained by our approach. The Type column shows the query ID and the fixability type that the query belongs to. The Ideal Recall column shows the percentage of results that could be correctly achieved by using the baseline method (a new query plan generated from scratch) and the Real Recall column shows the percentage of results that are actually correctly achieved by using our proposed method. For the two fully fixable versions (FF and CFF), we do not have any dead attributes, i.e., there must be other data sources that share the same usage as the crashed ones. As a result, the Ideal Recalls of these two versions are always 100%. For the two partial fixable versions (PF and CPF), the Ideal Recalls of these two versions are lower than 100% due to the existence of dead attributes. From Table 7.3 we can observe that for 18 out of 20 cases, the actual recalls achieved by our approach is exactly the same as the ideal recall that could be achieved by the baseline method. For the two cases where our method gives lower recalls (cases in bold font), these two queries ask for SNP ID information, and the data sources providing SNP ID crashed. Our method selects to use *Seattle* data source while the baseline method uses *dbSNP* data source. Because *Seattle* data source only covers partial SNP ID data, the final recall of our method is lower than the ideal version. This shows that in few cases, the query plan generated by our approach is not as good as the query plan generated by the baseline method. This is reasonable because our query plan is generated based on a subquery which means that our plan is a *local optimal*, but the baseline method generates plan from scratch using the original query, which means that the plans generated by the baseline method is the *global optimal*.
The above results show that although for a few cases, the plans from baseline method have better quality than the plans from our approach, for most of cases, our plans is as good as the plans from the baseline method. Furthermore, our approach has much faster query processing time. We further examine the correctness of the results from our method, and we found that query results for all experiment queries obtained from our incremental method are correct.

The above two experiments show that for most queries, our approach could obtain query results with quality as high as the baseline method (which finds the ideal results), and our approach outperforms the baseline method in terms of query answering time. This illustrates the effectiveness and efficiency of our approach.

![Figure 7.6: Adaptation Overhead](image)

**Figure 7.6: Adaptation Overhead**

**Method Overhead:** Figure 7.6 shows the ratio of the overhead of our approach as compared to the query answering time. The x-axis is the query IDs and the y-axis is the overhead ratio after multiplied by 10^6. The figure is plotted in Logarithmic scale. From this figure, we can see that the overhead for each of the 20 queries is less than 1% of the query answering time. This shows that our fault tolerance approach only causes little overhead to the system.
7.6 Summary

In this chapter, we proposed an incremental query plan generation approach to support fault tolerance for search queries over deep web data sources. We proposed novel data models for deep web data sources and data redundancy across related data sources. For a query plan with inaccessible data sources, we proposed an algorithm to find the minimal impacted subplan which needs to be replaced by a new query plan, and we developed an algorithm to generate the maximal fixable subquery from which the new partial plan could be generated based on a novel fixability model.

Our experiments show that our incremental query processing mechanism outperforms a baseline algorithm, in terms of the query answering time. We have also shown that the quality of the incremental query plans generated by our approach is almost as good as the quality of the plans generated by the baseline algorithm. Furthermore, the overhead of the our method is no more than 1% of the entire query answering time.
CHAPTER 8
EXTRACTING OUTPUT METADATA FROM SCIENTIFIC DEEP WEB DATA SOURCES

For a data integration system that integrates heterogeneous deep web data sources, discovering the metadata for each data source is the prerequisite for many other integration tasks. With rapidly increasing number of data sources, the metadata of deep web data sources must be obtained automatically, and not manually. For example, currently there are more than 1000 online databases in the biological domain and the number is still increasing rapidly every year [8]. In this chapter, we propose two model based approaches for automatically extracting metadata of deep web data sources.

8.1 Motivation

The metadata of a deep web data source that we consider is the input and output schemas of the data source. In the literature, there has been a lot of research on understanding input forms and extracting input schemas of deep web data sources [135, 101, 12]. For example, sub-figure (a) in Figure 8.1 shows the input query form of dbSNP, which is a biological data source providing data on Single Nucleotide Polymorphisms (SNPs). We can know that Reference cluster ID(rs#) is the attribute in the input schema of dbSNP.
However, a difficult and as yet unsolved problem is of extracting the output schema of scientific deep web data sources, where the results of a query do not always contain the complete set of output schema attributes. To further explain this problem, consider the following example. After a user submits an input query form, an output page corresponding to the input query is returned as the answer. The sub-figure (c) in Figure 8.1 shows a partial output page from dbSNP after specifying Reference cluster ID(rs#) to be rs7412 (the ID of a SNP). The labels that are in bold font and underlined are the output schema attributes. The values shown in italic font are the corresponding data values. If a data source returns values for all of its output schema attributes every time, the output schema can be extracted from the output page(s), using any of the many existing techniques [130, 134, 107, 52]. Unfortunately, many deep web data sources, especially in the scientific domain, only return a partial set of output schema attributes in response to a query, i.e., the ones that have non-NULL values for the particular input.

To further elaborate on this, we consider the following two examples.

**Motivating Example 1:** Let us reconsider the dbSNP data source, and consider
Table 8.1: Output Schema Coverage Statistics for 4 Biological Data Sources

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Total output metadata attributes</th>
<th>Maximum Coverage</th>
<th>Minimal Coverage</th>
<th>Average Coverage</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbSNP</td>
<td>79</td>
<td>75(95%)</td>
<td>41(52%)</td>
<td>58(73%)</td>
<td>11</td>
</tr>
<tr>
<td>Gene</td>
<td>59</td>
<td>47(80%)</td>
<td>14(24%)</td>
<td>35(59%)</td>
<td>12</td>
</tr>
<tr>
<td>JSNP</td>
<td>95</td>
<td>84(88%)</td>
<td>51(54%)</td>
<td>68(72%)</td>
<td>10</td>
</tr>
<tr>
<td>KEGG</td>
<td>28</td>
<td>22(79%)</td>
<td>14(50%)</td>
<td>18(64%)</td>
<td>4</td>
</tr>
</tbody>
</table>

Motivating Example 2: To further understand the complexity of this problem, we closely examined 20 data sources we had been working with. For each source, we randomly create 50 input cases, and analyze the corresponding 50 output pages. We count the number of distinct output schema attributes. Detailed results from 4 of these data sources is shown in Table 8.1. We can observe that none of the output pages covers the complete output schema, and on the average, each output page only covers about 60% to 70% of the complete output schema. Furthermore, the high standard deviation values also show the coverage of output schema varies considerably between different output pages.
8.1.1 Related Problems and Existing Work

The problem we are considering may appear similar to many problems that have been extensive studied. Among these, the existing output schema extraction methods [130, 134, 107, 52] focus on label extraction from HTML web pages. They have mostly been driven by data sources in the E-commerce domain, where most data sources return the complete set of output schema attributes for every input. In considering the data sources that return only a subset of output schema attributes, we need to intelligently obtain a list of output pages, so that these output pages could cover the complete output schema. In the process, the existing approaches cited above can be adopted to extract the output schema attribute labels.

Extracting data from deep web data sources is another problem somewhat related with ours [95, 41]. However, our problem is distinct from this problem from the following aspects. First, we focus on extracting metadata, not data. Second, the approaches proposed in [95, 41] assume that every output page contains the complete set of output schema attributes.

It may be argued that an intuitive way to find the output schema is to use the documentation from data sources. However, this approach is not applicable in many cases, and for other cases, there are other difficulties. For example, among the 20 biological data sources we focused on, only 10 had accessible documentations, and furthermore, the documentation that is available is hard to use. This is because most documentation is designed for human readers, and not for an automated system. For example, some data sources (dbSNP, Alfred) use non-text documents (such as an ER diagram) to state their schemas, while some other sources (Gene, KEGG\(^1\), Uniprot, \(\text{http://www.genome.jp/kegg/}\))

\(\text{http://www.genome.jp/kegg/}\)
GeneCards\(^2\) only provide a high-level schema information, without giving a complete list of attributes.

### 8.1.2 Our Approach

In this chapter, we propose two approaches for automatically extracting the set of output schema attributes from scientific deep web data sources. The first approach, which is called the *Sampling Model Approach*, is based on a distribution model of output schema attributes which we discovered from a preliminary study on scientific data sources. We argue that a modest sized sample of output pages could discover output schema attributes with relatively high recall. *Mixture Model Approach* is our second proposed approach. It is motivated by the observation that many scientific data sources share output schema attributes [15]. For example, SNP Frequency related schema attributes can be obtained through 3 data sources, which are dbSNP, SNP500Cancer, and SeattleSNP. Based on this characteristic of scientific deep web data sources, we consider that an output schema attribute is generated from a mixture model, which is composed of multiple related data source models.

### 8.2 Problem Formulation and Solution Overview

We have a set of deep web data sources \( D = \{D_1, D_2, \ldots, D_n\} \). Each data source \( D_i \) has an input schema \( IS^{D_i} = \{in_1^{D_i}, in_2^{D_i}, \ldots, in_k^{D_i}\}, k \geq 1 \). It also has an output schema \( OS^{D_i} = \{o_1^{D_i}, o_2^{D_i}, \ldots, o_m^{D_i}\}, m \geq 1 \), which we want to determine. An input instance for the data source \( D_i \) is a tuple with valid values for each input schema attributes. A naive way of obtaining the complete output schema of \( D_i \) is to use all possible input instances. However, this method is clearly infeasible because, first,

\(^2\)http://www.genecards.org/
it is very hard to obtain all possible input instances for a data source, and second, querying the data source using all input instances will be extremely time consuming. As a result, we need a more efficient approach.

**Problem Formulation:** For each data source $D_i$, we want to select $s_i$ number of input instances, where $s_i$ is the sample size. Each input instance $II^D_i$ will return an output page containing a partial output schema, $POS^D_i$. Then we will have $s_i$ partial output schemas for $D_i$, which can be denoted as $sampleOS^{D_i} = \{POS^D_1, POS^D_2, \ldots, POS^D_{s_i}\}$. Ultimately, for $D_i$, we want the union of all the partial output schemas to be equal to the complete output schema of $D_i$, denoted as $OS^{D_i} = \bigcup_{j=1}^{s_i} POS^D_j$.

Given the problem formulated above, for a data source, finding a list of output pages containing the *complete* output schema is the focus of our paper. When these output pages are obtained, we adopt existing methods to extract schema attribute labels from the page.

**Sampling Model Approach:** Since we take input samples to obtain a list of output pages instead of an exhaustive approach, we need to answer three questions: 1) what is the appropriate value of the sample size $s_i$?, 2) what is the possibility of missing some output schema attributes using a sample of this size?, and 3) where do we draw the samples from? These three questions are answered in Section 8.3. In the sampling model approach, we first establish a model to show that for deep web data sources, most output schema attributes can be discovered using a *modest* sized sample. Then we describe an input entity sample pool from which we draw our input samples, and propose a *rejection sampling* method to draw a simple random sample from the sample pool. We also estimate the sampling error and construct a sample size estimator.

**Mixture Model Approach:** In some cases, it is hard to obtain a sampling pool of the input instances for a data sources. In such cases, the sampling model approach
would not be applicable. Because the input interfaces of deep web data sources are
designed for human users, there are often input examples on the input interface that
can be extracted automatically. One such example is shown as a highlighted rectangle
in sub-figure (a) Figure 8.1. However, the number of input samples obtained in this
way would be too small to provide an effective approach. We can exploit another
observation, which is that there is likely some redundancy across data sources. Thus,
output schema attributes could be shared among different data sources. Since the
output page of the deep web data sources are dynamically generated from its back-end
database, we model a data source as a probabilistic data source model that generates
output schema attributes with certain probabilities. We also model the borrowability
between similar data sources. Since a schema attribute could be shared by multiple
data sources, we consider the probability of a schema attribute generated by a data
source $D$ as being determined by a mixture model composed of the probabilistic data
source models of similar data sources. Section 8.4 describes this method in more
details.

8.3 Sampling Model Approach

In this section, we introduce the sampling model approach. First, we empirically
build a distribution model for output schema attributes of deep web data sources, to
understand the underlying assumptions that can be valid. Then, we introduce the
rejection sampling method. Finally, we construct a sample size estimator to bound
the sampling error within a confidence interval.

8.3.1 Distribution Model

Given two randomly selected output pages from a deep web data source, we often
observe that some schema attributes only appear in one output page, but most schema
attributes appear on both. This observation points to an important fact about the relationship between output schema attributes and output pages. To further study this relationship, we downloaded 100 randomly selected output pages over 20 data sources we are targeting. Then, for each source, we draw a random sequence of output pages, and count the number of output schema attributes and the number of distinct output schema attributes, cumulatively. In Figure 8.2 we show the results for the data sources dbSNP, Gene and JSNP. The results for other data sources are similar. Here, the x-axis is the total number of output schema attributes seen. The y-axis is the number of distinct output schema attributes. We observe that with the increase of the number of output schema attributes, which also corresponds to the increase in the number of output pages, the number of newly discovered distinct output schema attributes reduces. In other words, as more output pages are gathered, there will be diminishing returns in terms of discovery of distinct output schema attributes.

The above observation makes a sampling approach feasible. It shows that a large percentage of output schema attributes can be discovered using a relatively small number of output pages. The pattern shown in Figure 8.2 is very similar to the Heaps law [9] in linguistics. Heaps law is an empirical law which describes the portion
of a vocabulary that is represented by an instance document. It states that as more instance text is gathered, there will be diminishing returns in terms of the discovery of the full vocabulary. Our data fits the Heaps law very well. If we use $V$ to represent the number of discovered distinct output schema attributes, and $m$ to show the number of output schema attributes, we will have $V = km^\beta$. We observe from our data (Figure 8.2) that, in the context of the deep web data we are working with, the parameter $k$ and $\beta$ are typically $5 \leq k \leq 6$ and $0.3 \leq \beta \leq 0.4$.

Another necessity for a sampling based technique is the availability of a sampling pool. Our work assumes that such a pool is available. In practice, many data sources use entity names as input schema attribute and a sample pool may arise from a query log of entity names associated with a data source, or could be generated using a domain-specific ontology. A problem with a sampling pool is that it may not have a complete list of valid input entities, which in reality is hard, if not impossible, to obtain. The under-coverage of the sampling pool may bring a non-sampling error. But we argue that the way our sampling pool is constructed decreases the effect of the non-sampling error. We first define a rare output schema attribute to be an attribute which seldom appear in output pages, formally, it is output in response to only a small number of potential inputs, these inputs are infrequently used in practice. The performance of a data integration system mainly depends on the non-rare output schema attributes. Since our sampling pool is constructed using a large number of frequently used inputs, we claim that it decreases the probability of missing non-rare output schema attributes, and thus the effect non-sampling error.

8.3.2 Sampling Algorithm

Simple random sample (SRS) is a widely used sample form which reduces bias in the sampling procedure [111]. SRS requires that each unit in the sampling pool should
have an equal probability of being sampled. Since our sample pool is composed of query logs, the more frequently used input instances are likely to appear more often. As a result, if we draw random samples from our sampling pool, the probability of a unit being sampled, which is defined as the selection probability, is proportional to its frequency. Therefore, an sample formed from a random walk on our sample pool could be skewed and the skew is resulted from the variance among the selection probabilities.

To counter the skew, we use rejection sampling. Rejection sampling probabilistically accepts or rejects the input instances that have been selected by the random walk on our sampling pool. Let’s consider the following example. We assume that the size of our sampling pool is $|Q|$, i.e. the total number of input instances in the sampling pool is $|Q|$. Consider an input instance $x$ appears $k$ times in the sampling pool and $x$ is selected from a random walk on our sampling pool. Ideally, since we want a simple random sample from the sampling pool, the probability of $x$ being sampled should be $\frac{1}{|Q|}$. But, actually the selection probability of $x$ will be $\frac{k}{|Q|}$. To compensate for the skew caused by the selection probability, our rejection sampling algorithm will toss a coin with a head probability of $\frac{1}{k}$, which is equal to the selection probability, and accepts $x$ if the head faces up. This decreases the possibility of $x$ being selected, and ensures the probability of $x$ being sampled is $\frac{k}{|Q|} \times \frac{1}{k} = \frac{1}{|Q|}$. A more detailed description of rejection sampling can be seen from the recent work by Bar-Yossef and Gurevich [10].

The rejection sampling method we use is illustrated as Algorithm 8.3.1.
Algorithm 8.3.1: RejectionSampling(Q)

while not enough samples
    randomly select an input case x from the sampling pool Q with replacement
    W=selection probability of x
toss a coin with a heads probability \( \frac{1}{W} \)
    if coin heads up
        accept sample x
    else reject sample x

Sampling Error: Using rejection sampling, we can draw a SRS from our sampling pool to estimate the number of output schema attribute V. Suppose the variance of the number of output schema attributes in the output pages of the input samples is \( s \) (the values in the last column in Table 8.1). The sampling error of our method is given by the lemma below.

**Lemma 8.3.1.** If the size of our sampling pool is \(|Q|\), we draw a sample of size \( n \), and the sampling error will be given by \( SE(V) = \sqrt{\frac{(|Q|-n)}{|Q|} \times \frac{s^2}{n}} \).

The proof of Lemma 8.3.1 is straightforward, using the properties of SRS [111]. The practical impact of the sampling and non-sampling errors will be shown in Section 8.5.2.

8.3.3 Sample Size Estimation

Given the sampling algorithm as in Section 8.3.2 and the sampling error as in Lemma 8.3.1, we need to determine the size of sample we need. Let \( V \) be the total number of distinct output schema attributes in a data source, and \( m \) be the total number of...
output schema attributes found in all the sampled output pages. From our distribution model, as we had introduced in Section 8.3.1, we know that \( V = km^\beta \). In order to estimate the sample size, we need to first compute the sampling error of \( V \) as shown in Lemma 8.3.1. To facilitate the computation, we make the following transformations. Let \( \bar{V} = \log(V) \), \( \bar{k} = \log(k) \), \( \bar{m} = \log(m) \), and \( \bar{\beta} = \beta \). Now our distribution model can be expressed as \( \bar{V} = \bar{k} + \bar{\beta} \times \bar{m} \).

**Lemma 8.3.2.** The sampling error of \( \bar{V} \) is \( SE(\bar{V}) = \sqrt{\frac{s^2_{\bar{V}}}{n}} \).

**Proof.** According to Lemma 8.3.1, we have:

\[
SE(V) = \sqrt{\frac{|Q| - n}{|Q|} \times \frac{s^2}{n}} \\
= \sqrt{\frac{|Q| - n}{|Q|} \times \frac{1}{n} \times \left( \frac{1}{n - 1} \sum_{i=1}^{n} (\bar{V}_i - E(\bar{V}))^2 \right)}
\]

In (2), \( |Q| \) is the total number of all possible output pages could be generated by the data source and \( n \) is the number of samples needed. We can know that \( |Q| \gg n \), as a result, \( \frac{|Q| - n}{|Q|} \to 1 \). \( SE(\bar{V}) \) can be written as:

\[
SE(\bar{V}) = \sqrt{\frac{1}{n} \times \left( \frac{1}{n - 1} \sum_{i=1}^{n} (\bar{V}_i - E(\bar{V}))^2 \right)}
\]

Now we need to compute the expectation of \( \bar{V} \) as follows:

\[
E(\bar{V}) = \frac{1}{n} \sum_{i=1}^{n} \bar{V}_i \\
= \frac{1}{n} (n\bar{k} + \bar{\beta} \sum_{i=1}^{n} \bar{m}) \\
= \bar{k} + \bar{\beta} \bar{m}
\]
By plugging (3) into (2), we can obtain the standard error of \( \tilde{V} \) as in (4):

\[
SE(\tilde{V}) = \sqrt{\frac{1}{n} \frac{1}{n-1} \sum_{i=1}^{n} (\hat{k} + \hat{\beta} \tilde{m}_i - \hat{k} - \hat{\beta} \bar{m}_i)^2}
= \sqrt{\frac{s_{\hat{m}}^2 \hat{\beta}^2}{n}}
\tag{4}
\]

To estimate the required sample size, for a variable \( Y \) needs to be estimated, we need to define its margin of error \( e \) and \((1 - \alpha)\%\) confidence interval \( CI \).

**Definition 8.3.3.** We have a variable \( Y \) needs to be estimated. The true value of \( Y \) is \( y \) and the estimated value is \( \hat{y} \). The margin of error \( e \) is defined as \( P(|y - \hat{y}| \leq e) = 1 - \alpha \). \( \alpha \) is the confidence of the estimation.

**Definition 8.3.4.** If the estimate of \( Y \), \( \hat{y} \), falls into the \((1 - \alpha)\%\) confidence interval of true value of \( Y \), we have \(|y - \hat{y}| \leq z_{\frac{\alpha}{2}} SE(Y)\).

Combining Definition 8.3.3 and Definition 8.3.4, for a variable \( Y \), we can relate the margin of error \( e \) of \( Y \) and the confidence interval \( CI \) of \( Y \) as follows:

\[
e(Y) = z_{\frac{\alpha}{2}} SE(Y)
\tag{1}
\]

In our scenario, the variable that needs to be estimated is \( \tilde{V} \). If we combine the formula (1) with Lemma 8.3.2, we can obtain

\[
e(\tilde{V}) = z_{\frac{\alpha}{2}} \sqrt{\frac{s_{\hat{m}}^2 \hat{\beta}^2}{n}}
\tag{2}
\]

**Lemma 8.3.5.** Let \( e_{\tilde{V}} \) be the error rate of \( \tilde{V} \), and \( s_{\hat{m}}^2 \) be the standard derivation \( \hat{m} \). The sample size needed in Algorithm 8.3.1 is

\[
n = \frac{s_{\hat{m}}^2 \times s_{\hat{m}}^2 \times \hat{\beta}^2}{e_{\tilde{V}}^2}
\]

**Proof.** Lemma 8.3.5 can be proved by solving \( n \) using the equation (2).
In statistical sampling, the confidence level $\alpha$ is usually set to be 0.05 [111], which makes $z_\alpha$ to be 1.96. In our system, $s_{n_0}^2$ is obtained through our experimental study that we had described earlier. As mentioned in Section 8.3.1, for the deep web data sources we are focusing on, the value of $\tilde{\beta}$ is usually set to be between 0.3 and 0.4. In Lemma 8.3.5, $\epsilon$ is only user specified parameter. In our system, we allow a margin of error of 5%, which means that we allow the number of estimated output schema attributes to be 5% lower than the real number of output schema attributes, denoted as $V_{\text{real}} = \frac{V_{\text{estimate}}}{0.95}$. According to the definition of margin of error in Definition 8.3.3, the margin of error is the deviation between the estimated value and the true value. As a result, we have $e(V) = V_{\text{real}} - V_{\text{estimate}} = \log\left(\frac{V_{\text{real}}}{V_{\text{estimate}}}\right) = \log\left(\frac{1}{0.95}\right) = 0.02$.

### 8.3.4 Obtaining output schema attributes

For a data source $D$, based on the sample size estimator we described above, we select a list of inputs with our sampling algorithm. These inputs are used to query the data source $D$ and obtain a list of output pages. Now, for each output page, we need to extract output schema attributes using an extraction algorithm. There are many existing methods [52, 101] can be used for extracting labels from HTML web pages, and in our implementation, we designed label classifier to extract schema attributes based on the ideas in [101]. For each output page of $D$, our classifier returns a schema attribute candidate set $SC$, where $SC = \{(sc_1, prob_1), (sc_2, prob_2), \ldots, (sc_n, prob_n)\}$. Each $sc_i$ is a schema attribute candidate and $prob_i$ indicates the probability that $sc_i$ is a true schema attribute in the output page. For each data source $D$, we can obtain a list of $SC$ sets, each of which is obtained from a sampled output page from $D$. Next, we merge all its $SC$ sets to obtain a final set of schema attribute candidates. During this merging, if a $sc_i$ has probability values from different output pages, we keep the the larger one for $sc_i$ for the final $SC$ set. After merging, we delete all the
schema attribute candidates with probability values smaller than a threshold. We notice that the classifier may bring new errors into our entire process. It is possible that a sampled output page contains a true output schema attribute, but the classifier misclassifies it. The effect of the classifier on the accuracy of our overall approach is shown in Section 8.5.2.

8.4 Mixture Model Approach

This section describes a different approach for extracting the output metadata of a deep web data source. The main idea in this approach is as follows. Let us say that for a data source $D$, an schema attribute $a$ does not appear in any of the output pages obtained from the sampled inputs we use. However, let us say that the schema attribute $a$ appears in the output of several other data sources, such that each of these data sources have many overlapping schema attributes with the data source $D$. In this case, there is a good chance that the schema attribute $a$ is in the output schema of the data source $D$, but has been missed because of the input sample we have used. In this case, we should try validating if $a$ does appear in output of $D$.

The rest of this section describes this approach in more details. We first introduce the model of data sources we use in this approach and then describe our method to measure the borrowability of a data source with respect to another. Finally, we introduce the validation algorithm we use.

8.4.1 Modeling of Data Sources

**Probabilistic Data Source Model:** The output page of a deep web data source is dynamically generated from its back-end database, based on the input. If we consider an output page as a document, then the data source can be considered as a collection of such documents, comprising all possible output pages that can be generated. As
each such document has a certain number of schema attributes, we can consider a
data source \( D_i \) as a probability model \( \Theta_i \) that generates schema attribute with certain
probabilities. The challenge for us is how to estimate this probability. For a data
source \( D \), if we have only one sample output page and the schema attribute candidates
are extracted using the classifier introduced in Section 8.3.4, an intuitive way to model
the probability is to use the probability score from the schema candidate set.

Formally, if our classifier predicts a schema attribute candidate \( sc_{ij} \) to be a true
schema attribute of a data source \( D_i \) with the probability \( \text{prob}(sc_{ij}, D_i) \), then the
probabilistic data source model \( \Theta \) says that the data source \( D_i \) generates schema
attribute \( sc_{ij} \) with the probability \( \text{prob}(sc_{ij}, D_i) \), which is denoted as \( p(sc_{ij} | \Theta_i) = \text{prob}(sc_{ij}, D_i) \).

**Mixture Model:** An important observation of scientific deep web data sources is
that many output schema attributes are common across distinct data sources. This
suggests that an output schema attribute could be generated from a **mixture model**
that is comprised of a set of probabilistic data source models. If a schema attribute
\( sc_{ij} \) is covered by the data source \( D_i \) with a probability \( \text{prob}(sc_{ij} | \Theta_i) \), and \( D_i \) has a set
of similar data sources \( D_1, D_2, \ldots, D_{i-1}, D_{i+1}, \ldots, D_m \), we consider the probabilistic
data source model of \( D_i \) as the **main model** and other similar data sources’ models as
**background models.** Along this line, the probability that a schema attribute candidate
\( sc_{ij} \) is a true schema attribute of data source \( D_i \) can be written as follows:

\[
p(sc_{ij} | D_i) = \text{prob}(sc_{ij} | \Theta_i) + \sum_{u=1,u\neq i}^{m} \lambda_{ui} \times \text{prob}(sc_{ij} | \Theta_u)
\]

The first component of the above expression represents the contribution from the
**main model** and the second component represents the contribution from the **background models.** \( \lambda_{ui} \) is the weight or **borrowability** given to the background model \( \Theta_u \)
with respect to main model \( \Theta_i \). Computation of this factor is explained next.

229
8.4.2 Computing Data Source Borrowability

In the discussion above, $\lambda_{ui}$ indicates the weight we put on the contribution from another data source $D_u$ with respect to $D_i$, which is also stated as the borrowability score of $D_u$ with respect to $D_i$.

The borrowability of $D_i$ with respect to $D_j$ is computed using their $SC$ sets, $SC_i$ and $SC_j$ which is the schema attribute candidate set returned by the label classifier as introduced in Section 8.3.4. We define the intersection of the two $SC$ sets from two data sources as $SC_i \cap SC_j = \{(sc_k, prob_k)|sc_k \in SC_i, sc_k \in SC_j, \text{prob}_k = \text{prob}_{SC_i}(sc_k) \times \text{prob}_{SC_j}(sc_k)\}$

In other words, the set $SC_i \cap SC_j$ contains the schema attribute candidates in both $D_i$ and $D_j$ and the probability is computed as the candidates appear in both sources. Then the borrowability of $D_i$ with respect to $D_j$ is defined as

$$\lambda_{ij} = \frac{\sum_{k=1}^{m} (\text{prob}_k|sc_k \in SC_i \cap SC_j)}{\sum_{i=1}^{n} (\text{prob}_i|sc_i \in SC_i)}$$

From this formula, we are more likely to borrow a schema attribute of $D_i$ to $D_j$ if more attributes in $D_i$ overlap with attributes in $D_j$.

Note that borrowability is not symmetric, i.e., the borrowability of $D_i$ with respect to $D_j$ may not be the same as the borrowability of $D_j$ with respect to $D_i$. For example, consider the situation where $D_i$ covers data about biological concepts SNP, Gene and Chromosome, and $D_j$ only focuses on SNP data. If we have a schema attribute about SNP, we should believe that the possibility of borrowing the attribute from $D_j$ to $D_i$ is higher than that of borrowing it from $D_i$ to $D_j$.

Another issue in borrowing schema labels from other data sources is that we need to match the labels from different data sources together to compute the set $SC_i \cap SC_j$. In our current system, we use some simple word level matching and instance based
matching methods to achieve this goal. For generalizing our work in the future, we will incorporate the more powerful methods proposed in the literature [61, 47].

Using the mixture data source model, for each data source $D$, a schema attribute candidate $sc$ has a new probability score, which is obtained by gathering contribution from multiple similar data sources, as the probability of $sc$ being a true attribute of $D$. If this probability is greater than a threshold, we predict it to be a true schema attribute of $D$.

### 8.4.3 Validation Of Borrowed Attributes

As we borrow output schema attributes from other data sources, we need to make sure that the borrowed attributes are not false-positives. Without having the access to the back-end databases, we use search engines and data source documentation to perform such validation. A reasonable assumption is that if a borrowed schema attribute is a true schema attribute of $D$, some documentation or HTML pages located within the domain of $D$ may contain this schema attribute. In our validation algorithm, for each borrowed schema attribute $ba$ with respect to the data source $D$, which has a corresponding web address URL, we issue a query ($Input, ba, URL$) on Google, where $Input$ includes the input schema attributes of $D$. If any documents or HTML pages are returned by the search engine, we believe that $ba$ is a true schema attribute of $D$.

### 8.5 Evaluation

This section reports the results from the experiments we conducted to evaluate our strategies. In our experiments, besides the 6 data sources (AlfredGene, AlfredSNP, dbSNP, Gene, SNP500 and KEGG) we have introduced in previous sections, we also considered 4 other deep web data sources in the biological domain, which are HGNC,
Initially, we report the performance of the label classifier and the validation heuristic we have used in our work. Quantifying the performance of these is important, as they impact the accuracy of our two proposed methods. Next, we compare the performance of two approaches proposed in this paper and point out the limitations of each approach. Finally, we show that a hybrid approach that combines the sampling model and the mixture model approach yields the best results.

8.5.1 Label Classifier and Validation Algorithm Evaluation

We use the precision (P) and recall (R) metrics to evaluate the performance of the classifier and the validation algorithm. Further, we report the F-measure, that incorporates both precision and recall with the following expression

\[ F = \frac{2PR}{P+R} \]

The results are shown in Table 8.2.

The performance of the classifier is shown in the left side of the table. For most data sources, our classifier has a F-measure about or above 0.8. This corresponds well to the results from the original paper from which our classification method was adapted [101]. The classifier does not perform well on the data source Gene. This is because unlike other data sources, which mainly use table-like format to represent the results, Gene uses text-like formats. The classifier misclassifies some labels in this case.

The validation algorithm works effectively on all data sources. All data sources have F-measure greater than 0.8 and only one data source has the F-measure lower

---

3http://snp.ims.u-tokyo.ac.jp/search.html#locus

4http://www.informatics.jax.org/javawi2/servlet

5http://www.informatics.jax.org/javawi2/servlet
Table 8.2: Precision and Recall of Label Classifier and Validation Algorithm

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Classify Precision</th>
<th>Classify Recall</th>
<th>Classify F-Measure</th>
<th>Validation Precision</th>
<th>Validation Recall</th>
<th>Validation F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alfred</td>
<td>0.99</td>
<td>0.87</td>
<td>0.93</td>
<td>1</td>
<td>0.93</td>
<td>0.96</td>
</tr>
<tr>
<td>dbSNP</td>
<td>0.80</td>
<td>0.94</td>
<td>0.86</td>
<td>0.87</td>
<td>0.76</td>
<td>0.81</td>
</tr>
<tr>
<td>Gene</td>
<td>0.70</td>
<td>0.35</td>
<td>0.47</td>
<td>1</td>
<td>0.74</td>
<td>0.85</td>
</tr>
<tr>
<td>HGNC</td>
<td>0.86</td>
<td>0.99</td>
<td>0.92</td>
<td>1</td>
<td>0.89</td>
<td>0.94</td>
</tr>
<tr>
<td>JSNP</td>
<td>0.93</td>
<td>0.65</td>
<td>0.77</td>
<td>1</td>
<td>0.9</td>
<td>0.95</td>
</tr>
<tr>
<td>MGI</td>
<td>0.88</td>
<td>0.70</td>
<td>0.78</td>
<td>1</td>
<td>0.75</td>
<td>0.86</td>
</tr>
<tr>
<td>SNP500</td>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>KEGG</td>
<td>1</td>
<td>0.75</td>
<td>0.86</td>
<td>1</td>
<td>0.72</td>
<td>0.85</td>
</tr>
<tr>
<td>Grand Average</td>
<td>0.87</td>
<td>0.75</td>
<td>0.79</td>
<td>0.99</td>
<td>0.84</td>
<td>0.90</td>
</tr>
</tbody>
</table>

than 0.85. We also observe that with the exception of dbSNP, the precision is 1 for all data sources. This indicates that our validation algorithm can recognize the true schema attributes with a high accuracy. In terms of the recall, we can observe that 5 data sources have high recall (greater than 0.85), whereas the other 4 have a moderate value (above 0.75). This is because these 4 data sources use some common English words, such as Method, Center and Resources, as schema attributes. These common words always appear in the documentation pages of the data sources, as a result, some non-attributes are accepted as false positives.
8.5.2 Evaluation Of Our Approaches

![Figure 8.3: Performance Comparison for Different Approaches: (a) Performance Comparison on dbSNP; (b) Performance Comparison on KEGG; (c) Performance Comparison on JSNP; (d) Performance Comparison on Gene.](image)

We now evaluate the performance of the two proposed approaches. As mentioned in Section 8.3.4, in the sampling model approach, the output schema attributes are extracted using a label classifier. As a result, the performance of the classifier may
negatively impact the performance of the sampling model approach. In order to separate the performance of the classifier with the sampling model approach, we assumed a perfect classifier just for the purpose of our experiments. This perfect classifier always has a recall and precision of 1.

Now, based on the two proposed approaches, we consider four different cases, which are as follows.

**SamplePC:** It refers to the sampling model approach with the perfect classifier. We use the sampling model approach to obtain the sampled output pages from data sources, and then we use the perfect classifier to extract schema attributes.

**SampleRC:** It refers to sampling model approach with the real classifier. We want to show the performance of the sampling model approach in a real data integration system and point out the limitations of the sampling model approach.

**Mixture:** It refers to the mixture model approach with the real classifier. In our experiments, we show the advantage of this algorithm and also point out the limitations.

**Sample+Mix:** It is a hybrid algorithm combining SampleRC and Mixture. In this algorithm, we first use the sampling model approach to select a list of sample output pages for each of the data sources. Next, we use the idea of the mixture model approach to share attributes, but with an important modification. In this hybrid algorithm, we not only share schema attributes across data sources, more importantly, we share schema attributes across different output pages from the same data source. Recall that the original mixture model approach assumes that for a data source $D$, the number of sample output pages is quite limited (usually 1). As a result, schema attribute sharing can only be performed among $D$ and its similar data sources. But in the hybrid algorithm, since we have multiple sample outputs from a data source $D$ obtained using input samples, we can consider each output page from
$D$ as a *pseudo-probabilistic data source model*. We extend the mixture model to not only include the data sources similar to $D$, but also all sampled output pages of $D$.

**Algorithm Performance and Analysis:** We report the detailed experiment results of the above four algorithms on the data sources dbSNP, KEGG, JSNP, and Gene. The precision and recall of the discovered output schema labels are used as the evaluation metrics. The results are shown in Figure 8.3. The experiment results of other 6 data sources we worked with are similar to the results shown here, and are not included because of space limitations. In the figures, the x-axis is the number of samples used, and the y-axis is the recall of the discovered schema attributes. The number near to each data point is the precision.

We have the following observations.

1). Using the SamplePC algorithm (line with a triangle pointing right), the recall increases rapidly with the increasing number of samples, but only till we have around 10 samples. Subsequently, as more samples are used, the rate of improvement of the recall slows down, and finally reaches about 95 – 97%. The pattern is about the same for all data sources. This pattern is consistent with the distribution model as we introduced in Section 8.3.1. We can see that with a perfect classifier, we can obtain almost all output schema attributes with a modest sized sample. For dbSNP, only 3% of the attributes are missed due to sampling and non-sampling errors, and this error rate (3%) is within the error rate specified in the sample size estimator (which is 5%). We can also see from the sub-figures (b), (c) and (d) that the percentage of missed attribute of the SamplePC algorithm is always below or about 5%. This shows that the sampling model approach achieves the specified error rate.

2). Using SampleRC algorithm (line with a triangle pointing down), the recall has the same pattern as in the SamplePC algorithm. However, the highest recall achieved is much lower than the recall achieved from SamplePC. This shows that in a real
system, the performance of the sampling model approach is degraded severely due to the performance of the classifier. Across the four data sources, the best results are obtained from dbSNP, where the highest recall is 90%. The results are the worst for Gene data source, because the classifier has poor performance, as we had shown earlier in Table 8.2. This observation shows an important limitation of the sampling model approach, which is that the accuracy of this approach, to some extent, depends on the accuracy of the label classifier.

3). To achieve the same recall with the Mixture model approach (line with circle), we need significantly fewer samples than the sampling model approach. For example, with dbSNP, only 4 samples are needed instead of 13. For KEGG data source (sub-figure (b)), the Mixture algorithm achieves a recall of 75% using 4 samples, but the highest recall of SampleRC is only about 70%. The same pattern also appears for JSNP and Gene. Another observation is that after the recall reaches some level, it does not change much with the increase in number of samples. Furthermore, the precision now decreases. The reasons are the follows. At the beginning of the Mixture algorithm, it successfully finds common schema attributes and shares them among different data sources. When all common schema attributes have been discovered, the algorithm cannot find more schema attributes. Since it is possible that the Mixture algorithm borrows false positives from other data sources, the precision goes down. This shows a limit to what can be achieved by the Mixture approach, which depends on the number of common schema attributes shared among data sources.

4). Using the Sample+Mix algorithm (line with square), for dbSNP (figure(a)), we can achieve a recall as high as the SamplePC algorithm, while using much less number of samples (10 v.s. 20). In the sub-figure (b), for KEGG, although the recall of Sample+Mix does not achieve the level achieved through SamplePC, the hybrid algorithm outperforms the SampleRC and Mixture algorithm. In the sub-figure (c), for JSNP,
the recall of the hybrid algorithm is even higher than the SamplePC algorithm. This shows that the hybrid algorithm overcomes the limitations of the sampling model approach and the mixture model approach.

The reasons that the hybrid algorithm outperforms the sampling and the mixture model approach are as follows. First, in the hybrid algorithm, we use the sampling approach to select more samples than in the mixture model approach, so the discovery of output schema attributes does not solely depend on the shared schema attributes across data sources. Second, hybrid algorithm can correct the misclassified non-rare attribute. For example, in a sample output page, a true schema attribute $sc$ is misclassified (has low probability score). If $sc$ appears multiple times in other output pages from the same data source (i.e. $sc$ is non-rare), since we also combine the contribution of different output pages in the hybrid algorithm, the gathered weighted probability for $sc$ would be higher than the cut-off threshold.

### 8.6 Summary

In this chapter, we considered the problem of extracting output metadata from deep web data sources. We proposed two approaches, which are the sampling model approach and the mixture model approach. Our experiments show that using a perfect classifier, the sampling model approach achieves high recall while not exceeding the pre-specified error rate. But, with a real classifier, the performance of the sampling model approach degraded. The mixture model approach achieves relatively high recall using significantly fewer samples. However, there is a limit to what can be achieved by this approach, which depends on the number of common attributes shared among data sources. A hybrid approach that combines the above two approaches gave the best results.
In the course of my research, I have noticed that the deep web is an increasingly important Web based information source which contains structured/semi-structured and topic/domain-specific data. However, the current research accomplishments in the deep web field are still far from satisfactory. In the future, I would like to explore my research in the following directions to find solutions for better understanding and querying the deep web data. We propose three potential research directions as future work.

9.1 Understanding the Deep Web Data

In the future, I will find solutions to better understand the deep web data from the following two aspects: data quality and data distribution.

1. Data Quality: The first aspect of data quality is data reliability. The authority, domain, and topic of a data source could impact the reliability of its data. If data in one data source is cited from or referred to the data in another data source, data provenance should also be considered. In the future, I will look into these problems to build deep web data reliability models. The second aspect of data quality refers to the correlation between data sources. Two data sources may contain overlapping or complementary data. With only limited access to the data, I will consider the
following problems: how we could find the relationships between the data in correlated data sources; and how the data correlation information can be used in query answering.

2. Data Distribution: In the future, I will consider the problem of learning the approximate data distribution from hidden data to aid approximate query answering over the deep web. Along this line, I need to address the following issues. First, with limited access to the data, what sampling method should be used and what the appropriate sample size should be so that we could learn data distribution with a bounded confidence interval. Second, since many data sources update their data frequently, how we can handle with the consistency issue.

9.2 Querying the Deep Web Data

In the future, I will consider the following issues related with querying the deep web data: deep web data source indexing, structured queries, semantic queries and result ranking.

1. Deep Web Data Source Indexing: Indexing is one of the most important considerations in any search system. Consider in a general setting, there are thousands of deep web data sources from multiple domains integrated in a deep web search system. Given a query, to identify the most relevant data sources efficiently, novel indexing methodologies specifically designed for the deep web are certainly needed.

2. Structured Queries: In the near future, I intend to continue to work on supporting structured queries over the deep web, such as correlated nested sub-queries, existence queries (Is-there style queries), and data mining queries. I believe that effective sampling algorithm is the core of addressing the above problems. I need to carefully consider what sampling algorithms are appropriate for these queries (e.g.
minimizing the estimation error propagation from sub-query to main query, appropriate sampling methods for “Is-there” estimation and data mining tasks such as ruling mining, classification and clustering, and adaptive methods to guarantee a confidence bound on data with limited accessibility).

3. Semantic Queries: I believe that to tackle the challenges inherent in deep web search, semantic web must be combined with the deep web. In the deep web, data is represented in heterogeneous types, such as text (maybe in different languages), audio, image, and table. Given a semantic query, i.e., a keyword query with semantic annotation or a query written in a semantic language, I intend to find solutions to the following problem: how a deep web search system could identify and extract semantically related data with different types. In the future, I will work on a semantic deep web search engine which could include the following tasks: deep web semantics annotation, (semi)automatic ontology generating from deep web, deep web semantic crawler, and semantic query answering.

4. Result Ranking: Returning ranked results to the user is important for a search system. However, result ranking for deep web query involves many challenges. For example, how to determine whether two answers returned from two correlated data sources are duplicates or near-duplicates? How to rank results in different representation formats, such as text, audio and image? In the future, I will work on result ranking algorithms based on data quality, ontology information and query/data semantics.
This section summarizes our contributions described in this thesis.

A popular and emerging trend in data dissemination involves online databases that are hidden behind query forms, thus forming the *deep web*. Hundreds of large, complex, and in many cases, related and/or overlapping, deep web data sources have become available. The popularity of this new data dissemination medium is creating new problems and challenges for data integration, data management and query answering. We have taken an important step in supporting automatic exploring and querying deep web data source by designing a data integration system SEEDEEP.

We contribute to the research area of data integrating and query answering over deep web data sources in the following aspects.

1. We study the problem of estimating the result of an aggregation query with low selectivity in presence of limited data access. To handle with low selectivity query, we use stratified random sampling. With limited data access, as in the deep web scenario, when the selective attribute is not directly queriable, as a result, we need to perform stratification on an auxiliary attribute which is queriable through the input query form. For this purpose, we proposed a Bayesian adaptive harmony search method to find the optimum stratification of the data. In this method, Bayesian analysis is used to adaptively update two
critical parameters of harmony search algorithm, and novel objective function and sampling allocation are proposed for handling with low selectivity queries.

2. We have developed two novel adaptive sampling algorithms, adaptive neighborhood sampling (ANS) and sub-space based two phase adaptive sampling (TPS), to approximately answer deep web aggregation queries which require data enumeration. Without any prior knowledge of hidden data’s distribution, our algorithms can obtain accurate estimations with low sampling costs.

3. We propose query planning algorithms for answering complex structured queries over deep web data sources. User could formulate a complex query into structured format which could includes aggregation, group by, value constraints, and nested queries. We proposed three query planning algorithms for different types of structured query. We also proposed several optimization techniques for speedup query execution.

4. We proposed a query-plan-driven query caching strategy for accelerating searches over deep web data sources. Our strategy involves not only caching previously extracted data, but more importantly, caching and utilizing the previously executed query plans. The proposed query reuse based query planning algorithm generates a detour query plan for a new query, which reuses cached query plans, and thus increases the possibility of reusing cached data.

5. We proposed an incremental query plan generation approach to support fault tolerance for search queries over deep web data sources. We proposed novel data models for deep web data sources and data redundancy across related data sources. For a query plan with inaccessible data sources, we proposed an algorithm to find the minimal impacted subplan which needs to be replaced by a new query plan, and we developed an algorithm to generate the maximal
fixable subquery from which the new partial plan could be generated based on a novel fixability model.

6. We considered the problem of extracting output metadata from deep web data sources. We proposed two approaches, which are the sampling model approach and the mixture model approach. A hybrid approach that combines the above two approaches extracts deep web data source output metadata with high recall with good precision.
BIBLIOGRAPHY


[73] Christian S. Jensen, Leo Mark, Nick Roussopoulos, and Timos Sellis. Using
differential techniques to efficiently support transaction time. The International

[74] Christopher Jermaine. Robust Estimation with Sampling and Approximate

Scalable approximate query processing with the dbo engine. In Proceedings of

[76] Ruoming Jin and Gagan Agrawal. A systematic approach for optimizing com-
plex mining tasks on multiple databases. In Proceedings of the 22nd Interna-
tional Conference on Data Engineering, page 17, 2006.

[77] Ruoming Jin, Leo Glimcher, Chris Jermaine, and Gagan Agrawal. New
sampling-based estimators for olap queries. In Proceedings of ICDE 2006,
page 18, 2006.

[78] Shantanu Joshi and Christopher Jermaine. Robust stratified sampling plans

[79] Shantanu Joshi and Christopher Jermaine. Sampling-Based Estimators for

[80] Navin Kabra and David J. Dewitt. Efficient mid-query re-optimization of sub-

[81] Varum Kacholia, Shashank Pandit, Soumen Chakrabarti, S.Sudarshan, Rushi
Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search
on graph databases. In Proceedings of the 31st international conference on Very
Large Data Bases, pages 505–516, 2005.

[82] Arthur M. Keller and Julie Basu. A predicate-based caching scheme for client-
server database architectures. The International Journal on Very Large Data

[83] K.E.Lohmanneller, C.L.Pearce, M.Pike, E.S.Lander, and J.N.Hirschhorn. Meta-
analysis of genetic association studies supports a contribution of common vari-

[84] Anastasios Kementsietsidis, Frank Neven, Dieter Van de Craen, and Stijn Van-
summeren. Scalable multi-query optimization for exploratory queries over fed-
erated scientific databases. Proceedings of the VLDB Endowment, 1:16–27,
2008.


254


