DATABASE USABILITY ENHANCEMENT IN DATA EXPLORATION

A Dissertation Presented

by

YUE WANG

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2017

College of Information and Computer Sciences

DATABASE USABILITY ENHANCEMENT IN DATA EXPLORATION

A Dissertation Presented

by

YUE WANG

Approved as to style and content by:

_____

Gerome Miklau, Co-chair

_____

Alexandra Meliou, Co-chair

_____

Prashant Shenoy, Member

_____

Tilman Wolf, Member

_____

James Allan, Chair of the Faculty
College of Information and Computer Sciences

DEDICATION

To my parents.

ACKNOWLEDGMENTS

This work would not have been possible without all my mentors and friends. First and foremost, I would like to express my deepest gratitude towards my advisors, Prof. Gerome Miklau and Prof. Alexandra Meliou. They are both brilliant researchers and patient teachers. They guided me in graduate research and taught me how to face life challenges. They have always been encouraging and supportive. I hope to be as successful and helpful as they are in my future career. I thank Prof. Prashant Shenoy and Prof. Tilman Wolf for serving on my committee and for their valuable suggestions.

I am truly grateful to my mentors. I thank my undergraduate advisor Prof. Shuigeng Zhou, who introduced me to the intriguing database research. I thank Dr. Haixun Wang for showing me how to do research at Microsoft Research Asia and opening a new world for me. I thank Dr. Lin Qiao whose words profoundly altered my attitude towards life during my internship at LinkedIn. I thank my mentor Dr. Yeye He when I interned at Microsoft Research. I was deeply impressed by his knowledge and learned a lot from him.

I would like to thank my friends, Yifei Huang, Bo Jiang, Tianzheng Wang, Wentao Wu, Pengyu Zhang, and Erkang Zhu. I will never forget the inspiring conversations with these intelligent, enthusiastic, and passionate researchers. I am fortunate to have them in my life.

Last but not least, I am grateful to the colleagues of the database group. It was a wonderful experience working with these bright and generous people. I am also grateful to my parents and all my extended family members for their constant love.

ABSTRACT

DATABASE USABILITY ENHANCEMENT IN DATA EXPLORATION

SEPTEMBER 2017

YUE WANG

B.Sc., FUDAN UNIVERSITY

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Gerome Miklau and Professor Alexandra Meliou

Database usability has become an important research topic over the last decade. In the early days, database management systems were maintained by sophisticated users like database administrators. Today, due to the availability of data and computing resources, more non-expert users are involved in database computation. From their point of view, database systems lack ease of use. So researchers believe that usability is as important as the performance and functionality of databases and therefore developed many techniques such as natural language interface to enhance the ease of use of databases. In this thesis, we find some deeper technical issues in database usability, so we look at several core database technologies to further improve the ease of use of databases in two dimensions: we help users process data and exploit computing capacities.

We start by helping users find the data. In the real world, public data is everywhere on the Web, but it is scattered around. We extract a prototype relational knowledge base to

solve this problem. We start from the most basic binary mapping relationships (sometimes also named bridge tables) between entities from the web. This mapping relationship facilitates many data transformation applications such as auto-correct, auto-fill, and auto-join.

After finding the data, we help users explore the data. When users issue queries to explore the data, their query results may contain too many items. So the system designer has to present a small subset of representative and diverse items rather than all items. This is known as the query result diversification problem. We propose the RC-Index, which helps to solve the diversification problem by significantly reducing the number of items that must be retrieved by the database to form a diverse set of a desired size. It is nearly an order of magnitude faster than the state-of-the-art and has a good performance guarantee, which improves the ease of use of databases in terms of querying.

Finally, we shift our focus from data to computing capacities. We propose a framework to help users choose configurations in the cloud. Cloud computing has revolutionized data analysis, but choosing the right configuration is challenging because the common pricing mechanism of the public cloud is too complicated. Users have to consider low-level resources to find the best plan for their computational tasks. To address this issue, we propose a new market-based framework for pricing computational tasks in the cloud. We introduce agents to help users configure their personalized databases, which improves the ease of use of databases in the cloud.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS . . . . . **v**

ABSTRACT . . . . . **vi**

LIST OF TABLES . . . . . **xiii**

LIST OF FIGURES . . . . . **xiv**

CHAPTER

1. INTRODUCTION **1**

1.1 Background 5

1.1.1 Mapping Synthesis 6
1.1.2 Query Result Diversification 7
1.1.3 Cloud Pricing 8

2. SYNTHESIZING MAPPING RELATIONSHIPS FOR DATA TRANSFORMATION **9**

2.1 Introduction 9

2.1.1 Solution Highlight 11

2.1.1.1 Synthesize Mapping Tables with Human Curation 11
2.1.1.2 Why Pre-Compute Mappings 11
2.1.1.3 Why Synthesize Tables 12
2.1.1.4 Existing Approaches 14
2.1.1.5 Contribution 15

2.2 Solution Overview 16

2.2.1 Preliminaries 16

2.2.1.1 Mapping Relationships 16

2.2.1.2 Table Corpora . . . 18
2.2.2 Solution Overview . . . 18
2.3 Candidate Table Extraction . . . 19
2.3.1 Column Filtering by PMI . . . 20
2.3.2 Column-Pair Filtering by FD . . . 21
2.4 Table Synthesis . . . 22
2.4.1 Compatibility of Candidate Tables . . . 22
2.4.1.1 Positive Evidence for Compatibility . . . 23
2.4.1.2 Negative Evidence for Incompatibility . . . 26
2.4.2 Problem Formulation for Synthesis . . . 28
2.4.2.1 Conflict Resolution . . . 35
2.4.2.2 Table Expansion . . . 37
2.4.3 Synthesized Mappings for Curation . . . 38
2.5 Experiments . . . 39
2.5.1 Experimental Setup . . . 39
2.5.1.1 Table Corpus . . . 39
2.5.1.2 Computing Environment . . . 39
2.5.1.3 Benchmarks . . . 40
2.5.1.4 Metrics . . . 41
2.5.1.5 Methods Compared . . . 41
2.5.2 Quality Comparison . . . 43
2.5.2.1 Detailed Example Mappings . . . 46
2.5.2.2 Usefulness of Mappings . . . 47
2.5.2.3 Individual Cases . . . 47
2.5.3 Run-time Comparison . . . 49
2.5.4 Sensitivity Analysis . . . 50
2.5.5 Experiments on `Enterprise` . . . 51
2.5.6 Effect of Conflict Resolution . . . 52
2.6 Related Work . . . 53

3. RC-INDEX: INDEX FOR RANGE QUERY RESULT DIVERSIFICATION 58

3.1 Introduction 58
3.2 Overview and Background 61

3.2.1 Result Diversification 62
3.2.2 Distance Function 64
3.2.3 Solution and System Overview 65
3.2.4 Cover Tree 67
3.2.5 Range Tree 69

3.3 Index-based Framework 70

3.3.1 Query Module: Sketch 70
3.3.2 Query Module: Diversity Index 72
3.3.3 Query Module: Range Index 73
3.3.4 Query Module: Candidate Extraction 74
3.3.5 Diversification Module 75

3.4 Performance Analysis 76

3.4.1 Query Quality and Complexity 76

3.4.1.1 Approximation Ratio 77
3.4.1.2 Time Complexity 80
3.4.1.3 Space Complexity 81

3.4.2 Index Complexity 81

3.4.2.1 Batch Construction 81
3.4.2.2 Insertion and Deletion 83
3.4.2.3 Space Complexity 86

3.5 Index Selection 87
3.6 MAXSUM 89
3.7 Experimental Evaluation 91

3.7.1 Settings 91

3.7.1.1 Data 91
3.7.1.2 Configuration 92
3.7.1.3 Comparison with Baselines 93

3.7.2 Quality and Scalability 94
3.7.3 Sensitivity 97
3.7.4 Data Stream 98

3.7.5 Index Selection 99

3.8 Related Work 99
3.9 Discussion 101

4. A CONSUMER-CENTRIC MARKET FOR DATABASE COMPUTATION IN THE CLOUD 103

4.1 Introduction 103
4.2 Computation Market Overview 107

4.2.1 Market Participants 107
4.2.2 Contracts 109
4.2.3 Properties and Assumptions 110

4.3 The Consumer's Point-of-View 110

4.3.1 Consumer Utility 111
4.3.2 Consumer Contract Proposal 112
4.3.3 Consumer's Contract Evaluation 114

4.4 The Agent's Point-of-View 115

4.4.1 Pricing Preliminaries 115
4.4.2 Contract Pricing 117

4.4.2.1 Linear Case 119
4.4.2.2 Selecting a Configuration 121

4.4.3 Risk-Aware Pricing 122

4.5 Fine-Grained Contract Pricing 123
4.6 Alternative Approaches 127

4.6.1 Benchmark-Based Approach 127
4.6.2 VCG-Auction-Based Approach 128
4.6.3 Differentiated Bertrand 130

4.7 Experimental Evaluation 135

4.7.1 Consumer Incentives 137
4.7.2 Agent Incentives and Market Properties 138

4.7.2.1 Competitiveness 138
4.7.2.2 Fairness 141
4.7.2.3 Resilience 144

4.7.3 Fine-Grained Pricing 145
4.7.4 Comparison with Alternative Approaches 146
4.7.4.1 Benchmark-based Approach 146
4.7.4.2 VCG-auction-based Approach 147
4.8 Related Work 148
4.9 Discussion 153
5. CONCLUSIONS **155**
BIBLIOGRAPHY **157**

LIST OF TABLES

Table **Page**

2.1 Example one-to-one mapping tables: (a) Countries to ISO codes, (b) Company names to stock-tickers, (c) State names to abbreviations, (d) Airports to IATA-codes. 10

2.2 Example many-to-one mapping tables: (a) Car makes and models, (b) Cities and states. 10

2.3 Auto-correction: correct inconsistent values (highlighted) using Table 2.1c. 10

2.4 Auto-fill: automatically populate values based on mappings from Table 2.2b. 10

2.5 Auto-join: joining related tuples based on mappings from Table 2.1b. 10

2.6 Examples from a synthesized mapping relationship (`country`, `country-ISO3-code`) using real web tables. The left table shows examples of synonyms for the country `South Korea`, all of which map to the same code `KOR`. The right table shows similar examples for `Congo`. 14

2.7 An example input table. Candidate two-column tables can be extracted using both PMI and FD filtering. 19

2.8 Example two-column binary tables for synthesis: (a) Countries and IOC codes, (b) Countries and IOC codes, where some countries use alternative synonyms compared to the first table, (c) Countries and ISO codes, where the code for some country can be different from the first two tables. 23

LIST OF FIGURES

Figure **Page**

2.1 Solution overview with three main steps: (1) Extract candidate two-column-tables; (2) Synthesize related tables; (3) Resolve conflicts in the same relationship. 16

2.2 Mappings from Wikipedia[1] for country names and three types of country codes: `IOC`, `FIFA`, and `ISO`. The three have identical codes for many countries, but also different ones for many others (in red circles). 23

2.3 Graph representation of candidate tables. Solid vertices on the left represent tables for ISO codes; hollow vertices on the right represent tables for IOC codes. Furthermore, solid edges indicate positive compatibility, while dashed edges indicate negative incompatibility. Edges with weight of 0 are omitted on the graph. 31

2.4 A real table with errors that can cause conflicts. 35

2.5 Example queries with "list of A and B" 40

2.6 Geocoding Systems 40

2.7 Average f-score, precision and recall comparison. 44

2.8 Additional mappings synthesized from `Web`. 47

2.9 Synthesized relationships not ideal as mappings. 47

2.10 Comparison with alternatives on individual cases (Sorted by f-score of our Synthesis approach). 48

2.11 Runtime. 50

2.12 Scalability. 50

2.13 Comparison with the alternative on `Enterprise` 51

2.14 Example mapping relationships and values, from the enterprise spreadsheets corpus 52

2.15 Conflict resolution improves performance. 53

3.1 Diversification on a sample of ATMs in New York City under two diversity score functions. We select $k = 10$ ATMs (red circles), out of a total of 30 (blue triangles) in each figure. The solution under f_{min} has better coverage than the one under f_{sum}. 64

3.2 Solution overview. Our approach involves two modules: Query Module and Diversification Module. 66

3.3 The highest four levels of an example cover tree on ATMs in New York City with normalized coordinates. Items at each level ℓ are highlighted in red. The tree satisfies Nesting, Covering, and Separation. 68

3.4 An example 2-dimensional range tree. Every node in the first dimension range tree points to another range tree in the second dimension. 69

3.5 RC-Index: maps every Range Index (RI) node to a Diversity Index (DI) tree. It is the core of Query Module. This example Range Index supports 1-dimensional range queries. 71

3.6 Performance. 76

3.7 The bound in Theorem 3.4.1 is tight in the worst case. 79

3.8 When $\mathbb{A}_q \supset \mathbb{A}_{RC}$, naïvely extend *RC* to answer q can result in arbitrarily bad diversity score. 88

3.9 Datasets. 92

3.10 Algorithms. 93

3.11 Synthetic data with varying size n: `RC-Index` is nearly an order of magnitude faster than the state-of-the-art while ensuring high quality result. 94

3.12 Insertion time of individual item increases as n grows. 95

3.13 Real-world data: `RC-Index` is the fastest approach while ensuring high quality result. 95

3.14 (a) Varying k: RC-Index delivers good result in short query time when k increases. (b) Varying d: RC-Index's query time gets closer to Greedy's but the result quality is the same as d increases. 96

3.15 Comparison under Manhattan distance. RC-Index outperforms the other algorithms. 97

3.16 (a) Varying b: greater b means longer query time. (b) Varying δ: greater δ means better result but longer runtime. 97

3.17 (a) Our algorithm works well on streaming data. (b) RC-Index supports partial range query well. 98

4.1 An overview of interactions of the main participants in the computation market: the consumer, the agent, and the cloud provider. 107

4.2 Utility function for Example 4.3.3 when $t < 20$. 113

4.3 (a) Price function for Example 4.3.5. (b) Comparison of two contracts. 114

4.4 An example of a simple relational query that can be broken into 3 subtasks, corresponding to different operators. 124

4.5 Types of Amazon machines and associated features and costs (in January 2015). The first 7 types (db.*) are RDS configurations, whereas the last 3 (m1.*) are EMR configurations. The prefixes (db and m1) are omitted from some figures for brevity. 135

4.6 (a, c) Users achieve better utility by using an expert agent, compared to naïvely selecting a default configuration. The agent benefits 40% of the consumers in RDS workloads, and 100% of the consumers in EMR workloads. (b, d) Expert agents always achieve the largest profits. This means that our market framework gives incentives to agents to find optimal configurations. 139

4.7 The expert agents select different configurations for different tasks to maximize profit. 140

4.8 Agents' estimates are often inaccurate, and such inaccuracies can lead to loss of profit. 141

4.9 Poor estimation moderately impacts the market. 142

4.10 By adjusting for risk, agents can reduce their losses in case of inaccurate estimates. 143

4.11 DP outperforms Greedy and Search 144

4.12 Pricing at finer granularities can vastly increase the agents' profits. 145

4.13 Consumers prefer our approach to benchmark except for highly repetitive workloads 146

4.14 VCG auction brings less profit to agents without necessarily reducing consumers' payments. (PP = our posted-price approach) 148

CHAPTER 1

INTRODUCTION

Since the invention of database management systems, databases have been used by sophisticated users like database administrators (DBAs) and professionals who process data within large institutions. These large institutions have very big data and complex requirements so they hire highly skilled DBAs to manage databases. For example, institutions do not have databases on their machines and clusters initially, so DBAs need to install and configure databases first. Institutions usually have limited computing resources, so DBAs must know how to assign disk space, memory, and CPU to databases and how to optimize the performance. Institutions care about fault tolerance, so DBAs back up data periodically. All the above tasks require the DBAs' expertise.

In contrast with the high level of expertise with these sophisticated users, unsophisticated users often outside large institutions have limited experience with databases but are increasingly involved in database computation. There are two reasons. First, data is more readily available and accessed by increasing number of non-experts. Today, data is produced by more applications and activities. For example, social networks and online shopping websites have captivated billions of users whose activities provide a large amount of data every day. The Internet of Things makes data collection easier and cheaper. Even an application developer may collect and analyze sensor data from wearable devices and smartphones. In addition, more data is publicly available. Many governments and organizations are publishing their data online, resulting in a "data democratization". For example, the federal government of the United States has established the "Data.Gov"[1] website con-

[1] https://www.data.gov

taining thousands of datasets to make the government more open and accountable. Another well-known example is the Sloan Digital Sky Survey[2] program that publishes terabytes of relational data through a Web interface. Microsoft even builds an Azure data market to connect data providers and users. Such ease of data collection and availability of data attract users with different technical backgrounds, many of whom are unfamiliar with databases. Second, computing capabilities offered by the public cloud brings availability of computing and cheap computing. Many large companies like Amazon, Google, and Microsoft are providing cloud services to end users. Every individual today can build a database on a virtual machine or cluster in the cloud. Cloud services are not expensive at all. For example, Amazon Web Service gives a user 750 machine hours per month for free. If the user runs out of the 750 machine hours, the cheapest virtual machine costs less than 1 cent per hour. In summary, the availability and affordability of data and capabilities make the user community much broader.

However, current database technology cannot support current users in all the above new scenarios in which data analysis may occur. Users still face many challenges if they want to utilize the available data and computing capacities. For instance, imagine an application developer Bob who is implementing a restaurant recommendation application. Although the Web is a rich resource of public data, he still has to browse the restaurant datasets online and manually examine their quality. Although the public cloud service providers make computational resources available to everyone, he is still confused about how to choose the appropriate virtual machine for his task. Although the database is good at answering various queries, he cannot find an easy way to query and present diverse restaurants in his application. Similar problems arise for a lot of non-expert users such as analysts and researchers from business and scientific fields. From their point of view, databases lack ease of use.

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

The lack of ease of use of databases is a hurdle between users and today's rich data/capacities, hindering non-expert users from performing data analysis. Researchers have already developed more advanced interfaces like natural language interfaces to improve database usability, but in addition to interfaces, we find some deeper technical issues that prohibit users from doing data analysis. In this thesis, we consider core database technologies that further enhance the ease of use of databases. We propose several new technologies in two dimensions. First, we help users process democratized data. Second, we help users exploit democratized computational capabilities.

In Chapter 2, we firstly help users find the data. In the real world, a relational dataset can be incomplete or incorrect. Although we can see public data everywhere on the Web, it is scattered around. For example, if a user wants to find a table of company names and stock tickers like ⟨Microsoft, MSFT⟩ in the global mainstream stock markets, it is very difficult to find a complete and accurate dataset. So the user has to search online and manually integrate this information from multiple websites. This process could be more painful when multiple websites inconsistently represent the same or equivalent entities. In this chapter, we extract a prototype relational knowledge base to solve this problem. We start from the most basic binary mapping relationships (sometimes also named bridge tables) between entities from the web. Given instances of mapping entities such as ⟨Microsoft, Redmond⟩ and a query "Facebook", we can return ⟨Facebook, Menlo Park⟩ as the result. Our algorithm solves two challenges: (1) how to extract meaningful individual mapping instances; (2) how to group them correctly. We compare our algorithm with existing alternative approaches. The evaluation shows that our algorithm has greater precision and recall.

In Chapter 3, we help users conveniently view the data through query result diversification. The problem of returning diversified query results consists of finding a small subset of valid query answers that are representative and different from one another, usually quantified by a diversity score. Query result diversification is widely used in data exploration,

Web search, and recommendation systems. Most existing techniques for query diversification first compute all query results and then find a diverse subset. They are inefficient when one or more users repeatedly query a substantial number of items in a database. In this chapter, we propose the RC-Index, which helps to solve the diversification problem by reducing the number of items that must be retrieved by the database to form a diverse set of a desired size. The RC-Index enables us to return a diverse set of, for example 25 items, from a set of a million items within one second. To the best of our knowledge, this is the first index-based method with guaranteed approximation ratio for range queries.

In Chapter 4, we improve the ease of use of publicly available computing capacity. The availability of public computing resources in the cloud makes data analysis easier, but requesting cloud resources often involves complex decisions for consumers. Estimating the completion time and cost of a computation and requesting the optimal cloud resources with respect to time and cost are challenging tasks even for an expert user. A suboptimal request of resources can make consumers miss the deadline or waste time or money. We propose a new market-based framework for pricing computational tasks in the cloud. Our framework introduces an agent between consumers and cloud providers. The agent takes data and computational tasks from users, estimates time and cost for evaluating the tasks, and returns to consumers contracts that specify the price and completion time. Our framework can be applied directly to existing cloud markets without altering the way cloud providers offer and price services. In addition, it simplifies cloud use for consumers by allowing them to compare contracts, rather than choose resources directly. We present design, analytical, and algorithmic contributions focusing on pricing computation contracts, analyzing their properties, and optimizing them in complex workflows. We conduct an experimental evaluation of our market framework over a real-world cloud service and demonstrate empirically that our market ensures three key properties: (a) that consumers benefit from using the market due to competitiveness among agents, (b) that agents have an incentive to price contracts fairly, and (c) that inaccuracies in estimates do not pose a significant risk

to agents' profits. Finally, we present a fine-grained pricing mechanism for complex workflows and show that it can increase agent profits by more than an order of magnitude in some cases.

In the following section we give a background discussion of this thesis.

1.1 Background

Databases play an important role in data analysis and data exploration. Many organizations and individuals deploy database management systems to collect data and perform data analysis because of the expressiveness of its relational model and its good performance. For example, many users query the database to look for patterns or trends interactively. They issue queries and get responses within a few seconds to make business decisions. These queries are known as analytical queries. When the data is small, even a spreadsheet application like Office Excel can do the work. When the data size reaches gigabytes or terabytes, only databases can answer queries efficiently thanks to many system optimizations. The success of analytical query evaluation on big data results from the good performance and functionality of databases.

However, performance and functionality alone are not enough to support data analysis. Researchers have noticed that the usability of databases is as important as its performance and functionality [83]. The study of usability dates back to 1970s. Early studies focus on improving the querying interface. For example, researchers study Query-By-Example [188], natural language query [142], form-based query [50, 33], and visual query [120]. Then more studies on database usability appear around 2000. For example, keyword search allows users to query the database with a set of keywords [39, 27, 184, 29]; Query relaxation returns results to users even when the user's query is imprecise [8, 151]. There are also studies on query recommendation, query explanation, query result visualization etc. Below we review some general concepts of three areas of this thesis: mapping synthesis, query result diversification, and cloud pricing.

1.1.1 Mapping Synthesis

Data cleaning [133] is a common step in data analysis. In practice, data is often incomplete, incorrect or inconsistent. For example, Li et al. [102] study the stock data from multiple data sources like Yahoo! Finance, Google Finance, NASDAQ, Bloomberg, and so on. They find that 83% data items are inconsistent, suggesting that real-world data may have many mistakes. So users must perform certain data transformation to make data usable. They can utilize various techniques including erroneous value detection [163], entity resolution [96, 62], schema matching [131], etc.

These techniques are not very user-friendly, so researchers have studied interactive techniques to further make data cleaning easier for users. For example, Potter's Wheel [135] allows users to gradually build transformations through graphical operations or examples. Ajax [56] helps users with interactive entity resolution. Wrangler [86] provides natural language descriptions and visual transform previews in addition to interactive data transformation.

Our mapping synthesis technique also attempts to facilitate data cleaning, with an emphasis on finding data for users. Specifically, we propose to extract and materialize mapping relationships for users. Mapping relationships are two-column tables like (zip-code, city-name) and (state, state-abbreviation) that satisfy functional dependency. These mappings enable rich applications in data cleaning. For example, if some items of a restaurant dataset miss city names but have zip codes, the mapping from zip code to city name can help users complete the missing city names. This is a typical auto-fill scenario in data cleaning. Another example scenario is auto-correct. When a relational dataset mixes state names and state abbreviations in a column, the mapping from state name to state abbreviation can fix the inconsistency.

1.1.2 Query Result Diversification

Diversification is a very common technique for visualizing query results, which enhances the ease of use of database. When a user queries the database, the query result may have too many answers to be visually displayed. Diversification helps the system select a representative diverse set of answers to present. For example, imagine a user who searches restaurants in New York City on Google Map. Google will return fewer than 30 restaurants at different locations, which is more informative than displaying all hundreds of restaurants or displaying a few restaurants in the same street. So query result diversification is widely used in Web search, data exploration, recommendation systems, and so on.

The diversification problem is an NP-hard optimization problem. A greedy polynomial algorithm can provide a 1/2 approximation ratio. This is known as the best ratio that can be achieved by polynomial algorithms unless P=NP [137]. However, this algorithm has to retrieve all answers of the query result first, which can be very slow when the query result covers millions of answers.

In order to shorten query evaluation time, we utilize indexes. When a table in the database is very large, it is prohibitively slow to scan the entire table to find the answers of a query. For example, given a table containing the salaries of employees, how can we find the first 10 employees with the most salary? Scanning thousands of employees is obviously unnecessary because the query result only has 10 answers. A better option is to build an index on salary attribute. This index maintains a collection of links pointing to the items and order the links according to the corresponding items' salary values. So we can follow the links to extract 10 answers one by one. Conventional indexes like B+ tree can answer ORDER-BY queries or range queries. Indexing is a central topic in databases and has been widely studied. In Chapter 3, we propose a novel index for query result diversification, which significantly reduces the evaluation time.

1.1.3 Cloud Pricing

Many researchers have attempted to improve the ease of use of cloud resources. Current pricing models of the mainstream cloud providers are at the resource level: they charge users based on how many resources the users consume. For example, Amazon RDS charges based on the capacity and number of computational nodes per hour. A more powerful node costs more per hour. This pricing mechanism guarantees profit for cloud providers but lacks ease of use for users, because users can hardly understand how many resources are required for their tasks. Floratou et al. [54] propose Benchmark as a Service to benchmark users' workload and suggest the optimal resource configuration for repetitive execution. Tanaka et al. [149] make cloud providers bid for service contracts under the VCG auction. Ortiz et al. [125, 126] propose to classify user tasks into service tiers like ($<$ 3.5 minutes, \$1.20/hour, SELECT 1 attribute FROM 1 table WHERE condition). All these pricing approaches make cloud computing more convenient for users. We will introduce how our mechanism solves some limitations of the existing approaches in Chapter 4.

CHAPTER 2

SYNTHESIZING MAPPING RELATIONSHIPS FOR DATA TRANSFORMATION

2.1 Introduction

Mapping tables, sometimes also referred to as *bridge tables* [94], are two-column tables where each distinct value in the left column maps to a unique value in the right column (or functional dependencies hold). Table 2.1 gives a few example mapping tables with one-to-one mapping relationships. Table 2.2 shows additional examples with many-to-one mappings.

Mapping tables like these are important data assets for a variety of applications such as data integration and data cleaning. We briefly discuss three scenarios here.

`Auto-correction`. Real-world tables are often dirty, where inconsistent values may be present in same columns. Table 2.3 shows such an example. The last column about `state` are mixed with both full state names and state abbreviations. An intelligent data quality agent, equipped with the mapping table in Table 2.1c, can easily detect and alert users about such inconsistency, by discovering that values in the left and right column of Table 2.1c are mixed in one user data column. Furthermore, it can automatically suggest corrections based on the mapping relationship (e.g., correcting `CA` to `California`).

`Auto-fill`. In this example scenario in Table 2.4, a user has a list of city names. She wants to add a column of state names corresponding to the cities. By just entering a few example values (e.g., `California` for `San Francisco`), the system automatically discovers the intent by matching existing value pairs with those in Table 2.2b, and can thus suggest to automatically fill remaining values in the right column (grayed out in Table 2.4).

Country	Code
United States	USA
Canada	CAN
South Korea	KOR
Japan	JPN
China	CHN
...	...

(a) ISO country codes

Ticker	Company
MSFT	Microsoft Corp
ORCL	Oracle
INTC	Intel
GE	General Electric
UPS	United Parcel Services
...	...

(b) Stock tickers

State	Abbrev.
Alabama	AL
Alaska	AK
Arizona	AZ
Arkansas	AR
California	CA
...	...

(c) State abbreviations

Airport Name	IATA
Los Angeles International Airport	LAX
San Francisco International Airport	SFO
Tokyo International Airport	HND
London Heathrow Airport	LHR
Beijing Capital International Airport	PEK
...	...

(d) Airport IATA codes

Table 2.1: Example one-to-one mapping tables: (a) Countries to ISO codes, (b) Company names to stock-tickers, (c) State names to abbreviations, (d) Airports to IATA-codes.

Model	Make
F-150	Ford
Mustang	Ford
Accord	Honda
Camry	Toyota
Charger	Dodge
...	...

(a) Car make and model

City	State
Chicago	Illinois
San Francisco	California
Los Angeles	California
Houston	Texas
Seattle	Washington
...	...

(b) City and state

Table 2.2: Example many-to-one mapping tables: (a) Car makes and models, (b) Cities and states.

ID	Employee	Residence State
2910	Bren, Steven	California
1923	Morris, Peggy	Washington
1928	Raynal, David	Oregon
2491	Crispin, Neal	CA
4850	Wells, William	WA
...	...	...

Table 2.3: Auto-correction: correct inconsistent values (highlighted) using Table 2.1c.

City	State
San Francisco	California
Seattle	Washington
Los Angeles	California
Houston	Texas
Denver	Colorado
...	...

Table 2.4: Auto-fill: automatically populate values based on mappings from Table 2.2b.

Ticker	Market Cap
GE	255.88B
WMT	212.13B
MSFT	380.15B
ORCL	255.88B
UPS	94.27B
...	...

Company	Total '89 -'13	Dem	Rep
General Electric	$59,456,031	41%	58%
Walmart	$47,497,295	52%	44%
Oracle	$34,216,308	35%	64%
Microsoft Corp.	$33,910,357	48%	50%
AT&T Inc.	$33,752,009	47%	51%
...	...	...	...

Table 2.5: Auto-join: joining related tuples based on mappings from Table 2.1b.

`Auto-join`. In data integration and ad-hoc data analysis, users often need to "join" two tables together, whose key columns may have different representations. In Table 2.5 for example, an analyst needs to join the left table that has stocks by their market capitalization, with the right table that lists companies by their political contributions, to analyze potential correlations. However a direct join is not possible since the subject column of the left table is stock tickers, while the right table uses company names. A system equipped with mapping tables would make the join possible by using Table 2.1b as an intermediate bridge that performs a three-way join to connect these two user tables, without asking users to provide an explicit mappings.

2.1.1 Solution Highlight

2.1.1.1 Synthesize Mapping Tables with Human Curation

In this work, we develop methods to automatically synthesize mapping relationships from existing table corpora, where the goal is to generate as many high-quality mappings as possible. Because algorithms are bound to make mistakes, additional human verification and curation can be used to ensure very high precision (Section 2.4.3). The resulting mappings can then be utilized to enable the applications discussed above in a unified manner.

2.1.1.2 Why Pre-Compute Mappings

While there are separate solutions for auto-join and auto-fill problems (e.g., [72, 176]), our approach has a few important advantages.

First, synthesized mappings are amenable to human inspection and curation, which is critical to ensure very high quality. In attempting to commercialize technologies similar to [72, 176] in enterprise spreadsheet software like Excel, the main feedback we received is the trustworthiness of results produced by black-box algorithms. Algorithms with even 99% correctness is still unacceptable in the context of enterprise spreadsheets, because any error introduced by algorithms would be difficult for users to detect, but is highly embarrassing and damaging in enterprise settings.

An analogy we would like to draw is the knowledge-bases used in search engines such as Google and Microsoft Bing. Similar to our problem, the quality required for knowledge-bases is also very high, so commercial knowledge bases are created in offline processes that combine algorithmic automation with human curation. Mapping tables can be viewed as the counterpart of knowledge bases in the relational world, where a similar curation process may be needed because of the quality requirement. And like search engines that have millions of users, spreadsheet software can reach millions of data analysts, such that the cost of curating mappings can be amortized over a large user base to make the effort worthwhile.

Second, synthesized mapping relationships can be materialized as tables, which are easy to index and efficient to scale to large problems. For example, instead of performing expensive online reasoning over large table corpora for specific applications like auto-join [72] and auto-fill [176], one could index synthesized mapping tables using hash-based techniques (e.g., bloom filters) for efficiently lookup based on value containment. Such logic is both simple to implement and easy to scale.

Lastly, mapping tables are versatile data assets with many applications. By solving this common underlying problem and producing mapping tables as something that can be easily plugged into other applications, it brings benefits to a broad class of applications as opposed to requiring separate reasoning logic to be developed for different applications (e.g., [72] for auto-join and [176] for auto-fill).

2.1.1.3 Why Synthesize Tables

Given table corpora such as HTML tables from web or spreadsheets from enterprises, fragments of useful mapping relationships exist. For example, the `country` and `country-ISO3-code` columns in Table 2.1a are often adjacent columns in same tables on the web. As such, an alternative class of approaches is to "search" tables based on input values and then ask users to select relevant ones (e.g., Google Web Tables [61], Microsoft Power Query [116],

and DataXFormer [1]). However, because desired values pairs often span across multiple tables, users frequently need to search, inspect and understand table results, before manually piecing them together from multiple tables. Our experience suggests that this process is often too cumbersome for end users.

Mappings synthesized from multiple tables, on the other hand, take away the complexity and make it easy for end users. More specifically, synthesized mappings have the following benefits.

• *Completeness*. In many cases one table only covers a small fraction of mappings in the same relationship. For example, while there exist thousands of airports, a web table like Table 2.1d often lists only a small fraction of popular airports. Stitching together tables in the same relationship provides better coverage and is clearly desirable.

• *Synonymous mentions*. Each individual table from a table corpus typically only has one mention for the same entity. For example, Table 2.1a has `South Korea` and `KOR`. In reality different tables use different but synonymous names. Table 2.6 shows real results synthesized from many web tables, which has different synonyms of `South Korea`. Similarly the right part has many synonyms for `Congo`. Note that a specific synonym of `South Korea` may not necessarily co-occur with another synonym of `Congo` in the same web table, and the probability of co-occurrence in conjunction with synonyms of additional countries is even lower. However, any combination of these synonyms may actually be used in user tables that may require auto-join or auto-fill. Using single tables as mappings would not provide sufficient coverage in these cases. On the other hand, if all these synonyms are synthesized together as one table like in Table 2.6, then any combination of these synonyms can still be covered without requiring users to perform manual synthesis from multiple tables.

• *Spurious mappings*. Certain mappings that appear to hold locally in single tables may not be meaningful. For example, a random table listing `departure-airport` and `arrival-airport` may happen to have values observe functional dependency at the in-

Country	Code
Korea (Republic)	KOR
Korea (South)	KOR
KOREA REPUBLIC OF	KOR
Korea, Republic of	KOR
Korea, Republic of (South Korea)	KOR
Korea, South	KOR
Republic of Korea	KOR
South Korea	KOR
...	...

Country	Code
Congo (Democratic Rep.)	COD
Congo (Demographic Republic of)	COD
Congo, Democratic Republic of the	COD
CONGO, DEMOCRATIC REPUBLIC OF (WAS ZAIRE)	COD
Congo, Democratic Republic of the (Congo & Kinshasa)	COD
Congo, The Democratic Republic of	COD
CONGO, THE DRC	COD
Democratic Republic of Congo	COD
...	...

Table 2.6: Examples from a synthesized mapping relationship (`country`, `country-ISO3-code`) using real web tables. The left table shows examples of synonyms for the country `South Korea`, all of which map to the same code `KOR`. The right table shows similar examples for `Congo`.

stance level. However, at a conceptual level this is not a useful mapping. Such a spurious mapping, when indexed from single tables, can trigger false-positive results for applications like auto-correct and auto-join. A holistic analysis of global relationships are necessary to identify true mappings from spurious ones.

2.1.1.4 Existing Approaches

Given that table synthesis is needed to assist human curation, we look at existing techniques that can be used here.

`Union tables.` Ling and Halevy et al. studied the problem of stitching together web tables in the same web-domain (where tables are more homogeneous) based on meta data such as column names [107]. While the technique is not designed to synthesize relationships from a large heterogeneous corpus, it is the only work we are aware of that performs table synthesis from corpora. We will show that adapting this to a large corpus of heterogeneous tables will fail, because column names are often undescriptive [37] that leads to over-grouping and low-quality mappings. For example, in Table 2.1a, the column name for countries are often just `name`, and the column name for country-codes may be `code`. As a result, grouping by column names tends to lump this table with other name-to-code mappings. Our approach reasons about compatibility of tables based on values, which are more reliable in telling the true relationships.

`Schema matching.` There is a long and fruitful line of research on schema matching that suggests possible mappings between table columns [132]. However, schema matching is typically used in database contexts for a small number of schemas, and produces pair-wise matches for human users to evaluate. In our problem we are given hundreds of millions of schemas as input, for which pairwise human verification is infeasible, and aggregation of pairwise decisions to a group level is necessary for human curation. Furthermore, since we are only interested in mapping relationships, which are a specific type of tables that always observe functional dependencies, we can derive additional *negative incompatibility* induced by FDs that is not explored by schema matching. For example, there are multiple country-to-code relationships such as (`country` $\rightarrow$ `ISO3-country-code`), (`country` $\rightarrow$ `FIFA-country-code`), (`country` $\rightarrow$ `IOC-country-code`), etc, all of which share substantial value overlap as well as similar column names. Schema matching techniques would identify them as matches and merge them incorrectly, whereas we would prevent the synthesis because of the FD-based incompatibility. Considering both positive and negative signals is critical for high-quality synthesis at a large scale.

`Knowledge base.` Knowledge bases (KB) such as Freebase [17] and YAGO [146] have important entity-relationships that can be viewed as synthesized (semi-automatically) from different sources. However, many mappings are missing from KB. For instance, YAGO has none of the example mappings listed in Table 2.1 (all of which are common mappings), while Freebase misses two (stocks and airports). Furthermore, for mappings that do exist in KB, they typically do not have synonyms like the ones in Table 2.6. Lastly, KB have limited coverage beyond the public web domain, such as mapping (`cost-center-name` $\rightarrow$ `cost-center-code`) that is specific to enterprises domains.

2.1.1.5 Contribution

Observing that mapping relationships are well-represented in tables, we propose to automatically synthesize mapping relationships using table corpora. We formalize this

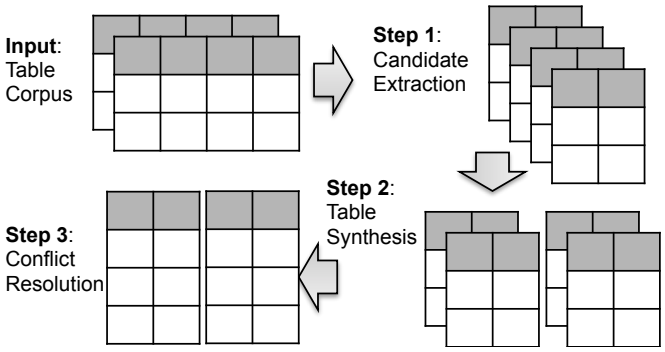


Figure 2.1: Solution overview with three main steps: (1) Extract candidate two-column-tables; (2) Synthesize related tables; (3) Resolve conflicts in the same relationship.

as an optimization problem that maximizes positive compatibility between tables while respecting constrains of negative compatibility imposed by functional dependencies. We show a trichotomy of complexity for the resulting optimization problem, and develop an efficient algorithm that can scale to large table corpus (e.g., 100M tables). Our evaluation using real table corpora suggests that the proposed approach can synthesize high quality mapping tables.

2.2 Solution Overview

In this section, we first introduce notions like mapping relationships and table corpora necessary for discussions. We then give a high-level overview of our synthesis solution.

2.2.1 Preliminaries

2.2.1.1 Mapping Relationships

. The goal of this work is to discover mapping relationships. Specifically, we focus on binary mappings involving two attributes.

Definition 2.2.1. *Let R be a conceptual relation with two attributes X, Y. The relationship is a mapping relationship, denoted by $M(X,Y)$ or $X \rightarrow Y$, if for all $x \in X$, x functionally determines one and precisely one value $y \in Y$.*

Examples of mapping relationships include (`country` → `country-code`) and (`company` → `stock-ticker`) as shown in Table 2.1 and Table 2.2. There is a mapping relationship between attributes `country` and `country-code`, for instance, since value in one attribute is uniquely associated with precisely one value in the other attribute.

Note that this is closely related to functional dependency (FD), traditionally defined over one physical table. We make the distinction to define mappings as conceptual relationships that can be represented in multiple tables, but may never be fully embodied in one physical table (e.g., the synthesized mapping shown in Table 2.6 with both `South Korea` and `Korea (South)` would not occur in one table).

Existing FD discovery work mainly focuses on efficiency (e.g., [79]), because it is intended for interactive data exploration on ad-hoc data sets. However, in our problem the key challenge is to produce high-quality synthesis of tables to assist human curation, where efficiency is not as important because the corpus is given a priori and synthesis can be run as offline jobs.

For cases where both $X \rightarrow Y$ and $Y \rightarrow X$ are mapping relationships, we call such bi-directional relationships 1:1 mappings (examples are in Table 2.1). If the mapping relationship only holds in one direction, then it is an N:1 mapping (Table 2.2).

It is worth noting that in practice, because of name ambiguity, functional relationship in some mappings may appear to only hold approximately. For example, `city` → `state` is conceptually a mapping relationship. However, when entities are represented as strings, the functional relationship may not completely hold. For example, in the same table there may be a city called `Portland` in the state of `Oregon`, and another city `Portland` in the state of `Maine`, thus giving the appearance of violating FD. To take such name ambiguity into account, we consider relationships whose surface forms are approximate mapping relationships.

Definition 2.2.2. *Let R be a conceptual relation with two attributes X, Y. The relationship is a θ-approximate mapping relationship, denoted by $M_\theta(X,Y)$ or $X \rightarrow_\theta Y$, if there exists a*

subset $\overline{R} \subset R$ with $|\overline{R}| \geq \theta|R|$, in which all $x \in X$ functionally determines one and precisely one value $y \in Y$.

We consider approximate mappings with θ over 95%. Hereafter we will simply use mapping relationship to refer to its θ-approximate version when the context is clear.

2.2.1.2 Table Corpora

The only input to our problem is a corpus of tables.

Definition 2.2.3. *A table corpus $\mathscr{T} = \{T\}$ is a set of relational tables T, each of which consists of a set of columns, or written as $T = \{C_1, C_2, ...\}$.*

Today relational tables are abundant and are very rich in nature. In this study, we use a corpus of 100M tables extracted from the Web, and a corpus of 500K tables extracted from spreadsheet files crawled from the intranet of a large enterprise.

2.2.2 Solution Overview

Our approach has three main steps, as shown in Figure 2.1.

• **Step 1: Candidate Extraction.** This step starts by exhaustively extracting pairs of columns from all tables in the corpus as candidates for synthesis. For each table $T = \{C_1, C_2, ..., C_n\}$ with n columns, we can extract $2\binom{n}{2}$ such ordered pairs. However, many column pairs are not good candidate for mapping relationships because (1) for some column pair if the local relationship is already not functional, then it is unlikely to participate in true mappings; and (2) some table columns are of low quality and are not coherent enough (e.g., with mixed concepts). To address these issues, we use FD constraints as well as value-based co-occurrence statistics to prune away low-quality candidate tables.

• **Step 2: Table Synthesis.** In this step, we judiciously synthesize two-column tables that describe the same relationship and are compatible with each other. The reason this is necessary is because many web tables and spreadsheets are for human consumption [107],

Home Team	Away Team	Date	Stadium	Location
Chicago Bears	Greenbay Packers	10-12	Soldier Field	Chicago, IL 60605
Detroit Lions	Minnesota Vikings	10-12	Ford Field	Detroit, MI
Detroit Lions	Greenbay Packers	10-19	Ford Field	Detroit, MI
Minnesota Vikings	Chicago Bears	10-19	US Bank Stadium	Minneapolis
Greenbay Packers	Minnesota Vikings	10-26	Lambeau Field	1265 Lombardi Ave
...	...	...	...	...

Table 2.7: An example input table. Candidate two-column tables can be extracted using both PMI and FD filtering.

and as a result contain only a subset of instances for the ease of browsing. Furthermore, one table in most cases mentions an entity by one name; synthesis helps to improve coverage of synonyms that are important for many applications.

• **Step 3: Conflict Resolution.** Because results from table synthesis piece together many tables, some of which are bound to have erroneous values inconsistent with others, namely two pairs of values in the same mapping with the same left-hand-side value but different right-hand-side (thus violating the definition of mappings). These can often happen due to quality issues or extraction errors. We apply a post-processing step to resolve conflicts in synthesized mapping relationships to produce our final results.

2.3 Candidate Table Extraction

In this section we briefly describe the preprocessing of tables. Recall that in this work we focus on synthesizing binary mapping relationships. We start with two-column tables extracted from an existing table corpus. Given a table $T = \{C_1, C_2, ...C_n\}$ with n columns, we can extract binary tables with pairs of columns $\{(C_i, C_j)|i, j \in [n], i \neq j\}$, for a total of $2\binom{n}{2}$ such column pairs. For example, in Figure 2.7, we can conceptually extract all pairs of columns such as (`Home Team`, `Away Team`), (`Home Team`, `Date`), (`Home Team`, `Stadium`), (`Home Team`, `Location`), etc.

Because not all these pairs are meaningful mappings, we filter out candidates with a coherence-based filtering and a local FD based filtering.

2.3.1 Column Filtering by PMI

When given a large table corpus (especially web tables), some tables are inevitably of low quality. Quality issues can arise because (1) columns may be mis-aligned due to extraction errors (especially for complicated tables like pivot table and composite columns); or (2) some table columns just have incoherent values.

In both of these cases, the resulting table column will appear to be "incoherent" when looking at all values in this column. For example, the last column `Location` in Table 2.7 have mixed and incoherent values. We would like to exclude such columns from consideration for mapping synthesis.

Therefore we measure the coherence of a table column based on semantic coherence between pairs of values. We apply a data-driven approach to define coherence based on co-occurrence statistics in a corpus. Let $s(u,v)$ be the coherence between two values u and v. Define $\mathscr{C}(u) = \{C | u \in C, C \in T, T \in \mathscr{T}\}$ as the columns in the table corpus $\mathscr{T}$ containing value u, and define $\mathscr{C}(v)$ similarly. Clearly, if $\mathscr{C}(u) \cap \mathscr{C}(v)$ is a large set, it means u and v are co-occurring frequently (e.g., $u =$ `USA` and $v =$ `Canada`). Then they intuitively are highly related and thus should have a high semantic coherence score.

We use Point-wise Mutual Information (PMI) [35] to quantify the strength of co-occurrence as a proxy for coherence.

$$\text{PMI}(u,v) = \log \frac{p(u,v)}{p(u)p(v)} \tag{2.1}$$

Where $p(u)$ and $p(v)$ are the probabilities of seeing u and v from a total of N columns in a table corpus $\mathscr{T}$, defined as $p(u) = \frac{|\mathscr{C}(u)|}{N}$, $p(v) = \frac{|\mathscr{C}(v)|}{N}$ and $p(u,v) = \frac{|\mathscr{C}(u) \cap \mathscr{C}(v)|}{N}$.

Example 2.3.1. *Let $u =$ `USA` and $v =$ `Canada`. Suppose $N = 100M$ (there are a total of 100M columns), $|\mathscr{C}(u)| = 1000$, $|\mathscr{C}(v)| = 500$, and $|\mathscr{C}(u) \cap \mathscr{C}(v)| = 300$ (individually, the two strings occur 1000 and 500 times respectively; together they co-occur 300 times). It can be calculated that $PMI(u,v) = 4.78 > 0$, suggesting that they have high co-occurrence and strong semantic coherence.*

We define coherence of two values, denoted by $s(u,v)$, as a normalized version of PMI called Normalized PMI (NPMI), which has a range of $[-1,1]$:

$$s(u,v) = \text{NPMI}(u,v) = \frac{\text{PMI}(u,v)}{-\log p(u,v)}$$

Using $s(u,v)$, the *coherence score* of a column $C = \{v_1, v_2, ...\}$, denoted as $S(C)$, is simply the average of all pair-wise scores.

$$S(C) = \frac{\sum_{v_i, v_j \in C, i<j} s(v_i, v_j)}{\binom{|C|}{2}} \tag{2.2}$$

We can then filter out a column C if its coherence $S(C)$ is lower than a threshold.

Example 2.3.2. *Table 2.7 is an example table with five columns. Column coherence computed using NPMI in Equation* (2.2) *would reveal that the first four columns all have high coherence scores, because values in these columns co-occur often in the table corpus.*

The last column `Location`*, however, has low coherence, because values in this column are mixed and do not co-occur often enough in other columns. We will remove this column when generating column pairs.*

2.3.2 Column-Pair Filtering by FD

After removing individual columns with low coherence scores, we use the resulting table $T = \{C_1, C_2, ..., C_n\}$ to generate binary tables with ordered column pairs $B(T) = \{(C_i, C_j) |\ i, j \in [n], i \neq j\}$ as candidate tables. However, most of these two-column tables do not express meaningful mapping relationships, such as (`Home Team`, `Away Team`), and (`Home Team`, `Date`) in Table 2.7.

Since our goal is to produce mapping relationships, we apply local FD checking to prune away column pairs unlikely to be mappings. As discussed in Definition 2.2.3 we account for name ambiguity (like (`Portland` $\rightarrow$ `Oregon`) and (`Portland` $\rightarrow$ `Maine`)) by allowing approximate FD that holds for 95% of values.

Example 2.3.3. *Continue with Example 2.3.2, we have pruned away the last column* `Location` *from Table 2.7 based on coherence scores. Four columns remain, for a total of* $2\binom{4}{2} = 12$ *ordered column pairs. Only* 2 *out of the* 12 *column pairs satisfy FD, namely, (*`Home Team`*,* `Stadium`*) and (*`Stadium`*,* `Home Team`*).*

We note that around 78% candidates can be filtered out with these methods. Algorithm 1 gives the pseudo-code for candidate table extraction. The two steps correspond to PMI-based filtering and FD-based filtering, respectively.

Algorithm 1: Candiate Extraction

Input: Table corpus $\mathscr{T}$
Output: Candidate two-column table set $\mathscr{B}$

```
1  B ⇐ ∅
2  foreach T ∈ T do
3      T' ⇐ ∅
4      foreach C_i ∈ T do
5          if C_i is not removed by PMI filter then
6              T' ⇐ T' ∪ {C_i}
7      foreach C_i, C_j ∈ T' (i ≠ j) do
8          B ⇐ (C_i, C_j)
9          if B is not removed by FD filter then
10             B ⇐ B ∪ {B}
```

2.4 Table Synthesis

Using candidate two-column tables produced from the previous step, we are now ready to synthesize relationships. Recall that synthesis provides better coverage for instances (e.g., synonyms) as discussed in the introduction.

2.4.1 Compatibility of Candidate Tables

In order to decide what candidate tables should be stitched together and what should not, we need to reason about compatibility between tables.

Flag	Country	IOC	FIFA	ISO
	Afghanistan	AFG	AFG	AFG
	Albania	ALB	ALB	ALB
	Algeria	ALG	ALG	DZA
	American Samoa[1]	ASA	ASA	ASM
	Andorra	AND	AND	AND
	Angola	ANG	ANG	AGO
	Antigua and Barbuda	ANT	ATG	ATG
	Argentina	ARG	ARG	ARG
	Armenia	ARM	ARM	ARM
	Aruba	ARU	ARU	ABW

Figure 2.2: Mappings from Wikipedia[1] for country names and three types of country codes: `IOC`, `FIFA`, and `ISO`. The three have identical codes for many countries, but also different ones for many others (in red circles).

Country	IOC
Afghanistan	AFG
Albania	ALB
Algeria	ALG
American Samoa	ASA
South Korea	KOR
US Virgin Islands	ISV

(a) B_1: IOC-(1)

Country	IOC
Afghanistan	AFG
Albania	ALB
Algeria	ALG
American Samoa (US)	ASA
Korea, Republic of (South)	KOR
United States Virgin Islands	ISV

(b) B_2: IOC-(2)

Country	ISO
Afghanistan	AFG
Albania	ALB
Algeria	DZA
American Samoa	ASM
South Korea	KOR
US Virgin Islands	VIR

(c) B_3: ISO

Table 2.8: Example two-column binary tables for synthesis: (a) Countries and IOC codes, (b) Countries and IOC codes, where some countries use alternative synonyms compared to the first table, (c) Countries and ISO codes, where the code for some country can be different from the first two tables.

2.4.1.1 Positive Evidence for Compatibility

Let $B = \{(l_i, r_i)\}$ and $B' = \{(l'_i, r'_i)\}$ be two binary relationships produced by the previous step, each with sets of (left, right) value pairs. If these two relations share many common value pairs, or $|B \cap B'|$ is large, they are likely in the same relationship and compatible for synthesis.

Let $w^+(B, B')$ be the *positive compatibility* between B and B'. We would like to use set-based similarity to quantify compatibility based on the overlap $|B \cap B'|$. However, common metrics like Jaccard Similarity, defined as $\frac{|B \cap B'|}{|B \cup B'|}$, would not work because if one small rela-

[1] `https://en.wikipedia.org/wiki/Comparison_of_IOC,_FIFA,_and_ISO_3166_country_codes`

tion is fully contained by another ($B \supset B'$, $|B| \gg |B'|$), the compatibility should intuitively be high, but the Jaccard Similarity score would actually be low.

Containment metrics would mitigate this issue, but Jaccard Containment is asymmetric – we want it to be symmetric because both the compatibility of B, B' and the compatibility of B', B are essentially the same thing ($w^+(B,B') = w^+(B',B)$). Given these we use a symmetric variant of Jaccard Containment called *Maximum-of-Containment* [19] for $w^+(B,B')$:

$$w^+(B,B') = \max\{\frac{|B \cap B'|}{|B|}, \frac{|B \cap B'|}{|B'|}\} \tag{2.3}$$

Example 2.4.1. *Table 2.8 shows three two-column candidate tables, B_1, B_2 and B_3, respectively. The first two are for the `IOC` code, while the last is for a different `ISO` code. All of these three are valid mappings but are for two different country-code standards, as explained in Figure 2.2.*

Using Equation (2.3)*, we can compute the positive compatibility between each pair of tables. For example, we have $w^+(B_1,B_2) = \max\{\frac{3}{6}, \frac{3}{6}\} = 0.5$, because $|B_1 \cap B_2| = 3$ (the first three rows), suggesting that the two tables share a significant fraction of mappings and are likely to be compatible for synthesis.*

Efficiency. Although conceptually compatibility scores can be computed for all pairs of candidates, in reality most tables share no common values, and will have a score of 0. A practical issue here is that given N total candidate tables, we need to perform $O(N^2)$ expensive containment computations. With millions of tables, this quadratic step is too expensive even for large Map-Reduce clusters.

In reality we observe that the scores for most pairs of tables are zero since they share no overlapping values at all. For example, Table 2.1a is about countries and Table 2.1b is about stock tickers. They have no overlaps in value-pairs, so both positive and negative weights are 0. Computing scores for these non-overlapping sets is clearly wasteful.

To address this problem, we use inverted-index-like regrouping in a Map-Reduce round to map all tables sharing at least some common value-pairs to the same partition, so that

compatibility is computed only for pairs of tables within each partition. Specifically, we evaluate $w^+(B,B')$ only if B and B' share more than $\theta_{overlap}$ value pairs (both left and right values), and similarly we evaluate $w^-(B,B')$ only if B and B' share more than $\theta_{overlap}$ left-hand-side values. In practice, the number of non-zero weighted edges is much smaller than N^2. This optimization makes it possible to scale the pair-wise computation step to hundreds of millions of tables.

Approximate String Matching: In real tables, values from different tables often have slight variations, such as "Korea, Republic of" & "Korea Republic", or "American Samoa" & "American Samoa (US)". In practice, there are other extraneous information in table cells, such as the footnote mark "[1]" in the fourth row in Figure 2.2. These artificially reduce positive compatibility and in some cases increase negative compatibility between tables, which is undesirable.

To account for such minor syntactic variations, we use approximate string matching between cell values. Specifically, we measure the Edit Distance, denoted as $d_{ed}(v_1,v_2)$, between a pair of values v_1 and v_2. We treat v_1 and v_2 as a match if $d_{ed}(v_1,v_2)$ is smaller than a threshold θ_{ed}. Here we use a fractional threshold defined as $\theta_{ed} = \min\{\lfloor|v_1|\cdot f_{ed}\rfloor, \lfloor|v_2|\cdot f_{ed}\rfloor\}$, which is dynamically determined based on the length of string $|v_1|$, $|v_2|$, and a fixed fractional value f_{ed} (e.g., 0.2). We choose to use a fractional distance instead of an absolute distance, because the desired edit distance should change based on the length of values. For example, for short values such as "USA" or "RSA" (for South Africa), any absolute distance threshold ≥ 1 would incorrectly match the two. Fractional threshold on the other hand would require an *exact* match for short strings like these. We further restrict the threshold to be within some fixed threshold $k_{ed} = 10$ to safeguard false positives. Combining, we use $\theta_{ed}(v_1,v_2) = \min\{\lfloor|v_1|\cdot f_{ed}\rfloor, \lfloor|v_2|\cdot f_{ed}\rfloor, k_{ed}\}$.

Example 2.4.2. *We continue with Example 2.4.1 in Table 2.8. When using approximate matching for positive compatibility,* $w^+(B_1,B_2)$ *will now be updated to* $\max\{\frac{4}{6},\frac{4}{6}\} = 0.67$. *This is because in addition to the first three matching rows between* B_1 *and* B_2*, now the*

fourth row "American Samoa" and "American Samoa (US)" will also be considered as a match, as the Edit Distance between the two values is 2 (ignoring punctuations), which is no greater than $\theta_{ed} = \min\{\lfloor 13 \cdot 0.2 \rfloor, \lfloor 15 \cdot 0.2 \rfloor, 10\} = 2$.

Efficiency. There are hundreds of millions of table pairs for which we need to compute compatibility. Let m and n be the numbers of values in a pair of tables. For each pair we need to make $O(nm)$ approximate string comparisons, each of which is in turn $O(|v_1||v_2|)$ when using conventional dynamic programming on the full matrix. This is too expensive even for production Map-Reduce clusters.

Our observation is that the required edit distance threshold θ_{ed} is small in most cases. So using ideas similar to the Ukkonen's algorithm [153], we only compute DP on the narrow band in the diagonal direction of the matrix, which makes it $O(\theta_{ed} \cdot \min\{|v_1|, |v_2|\})$. Since θ_{ed} is small it makes this step feasible. The algorithm for efficient approximate string matching is shown in Algorithm 2. We leverage the fact that the desired distance θ_{ed} is often small to only perform dynamic programming on a narrow band in the diagonal direction of the matrix instead of performing a full DP, which is in spirit similar to Ukkonen's algorithm [153].

Synonyms: In some cases, synonyms of entity names may be available, e.g., using existing synonym feeds such as [26]. If we know, for instance, "US Virgin Islands" and "United States Virgin Islands" are synonyms from external sources, we can boost positive compatibility between B_1 and B_2 in Table 2.8 accordingly.

2.4.1.2 Negative Evidence for Incompatibility

Positive evidence alone is often not sufficient to fully capture compatibility between tables, as tables of different relationships may sometimes have substantial overlap. For example, it can be computed that the positive compatibility between B_1 in Table 2.8a and B_3 Table 2.8c is $\max\{\frac{3}{6}, \frac{3}{6}\} = 0.5$ (the first, second and fifth rows match). Given the high score, the two will likely merge incorrectly (note that one is for `IOC` code while the other is

Algorithm 2: Approximate String Matching

Input: Strings v_1 and v_2, distance bound θ_{ed}
Output: Boolean $Matched$

```
if |v1| > |v2| then
    swap(v1, v2)
dist_{|v1|,|v2|} ⇐ ∞
dist_{i,0} ⇐ i,    ∀(i ∈ 0..|v1|)
dist_{0,j} ⇐ j,    ∀(j ∈ 0..|v2|)
for i ∈ 1..|v1| do
    lower ⇐ max{1, i − θ_edit}
    upper ⇐ min{|v2|, i + θ_edit}
    for j ∈ lower..upper do
        dist_{i,j} ⇐ ∞
        if dist_{i−1,j} ≠ NULL then
            dist_{i,j} ⇐ min{dist_{i−1,j} + 1, dist_{i,j}}
        if dist_{i,j−1} ≠ NULL then
            dist_{i,j} ⇐ min{dist_{i,j−1} + 1, dist_{i,j}}
        if dist_{i−1,j−1} ≠ NULL then
            dist_{i,j} ⇐ min{dist_{i−1,j−1} + 𝟙{v1[i] ≠ v2[j]}, dist_{i,j}}
Matched ⇐ (dist_{|v1|,|v2|} ≤ θ_ed)
```

for `ISO`). This issue exists in general when one of the columns is short and ambiguous (e.g. codes), or when one of the tables has mixed values from different mappings (e.g., both city to state and city to country).

We observe that in these cases the two tables actually also contain *conflicting* value pairs, such as the third and fourth row in the example above where the two tables have the same left-hand-side value, but different right-hand-side values. This violates the definition of mapping relationship, and is a clear indication that the two tables are not compatible, despite their positive scores.

We thus introduce a negative *incompatibility* between tables. Given two tables B and B', define their *conflict set* as $F(B,B') = \{l|(l,r) \in B, (l,r') \in B', r \neq r'\}$, or the set of values that share the same left-hand-side but not the right-hand-side. For example, between B_1 in Table 2.8a and B_3 Table 2.8c, (Algeria, ALG) and (Algeria, DZA) is a conflict.

To model the (symmetric) incompatibility between two tables B and B', we define a negative incompatibility score $w^-(B,B')$ similar to positive compatibility in Equation (2.3):

$$w^-(B,B') = -\max\{\frac{|F(B,B')|}{|B|}, \frac{|F(B,B')|}{|B'|}\} \tag{2.4}$$

Example 2.4.3. *We continue with Example 2.4.2 in Table 2.8. As discussed earlier, the positive compatibility between B_1 in Table 2.8a and B_3 in Table 2.8c is $\max\{\frac{3}{6}, \frac{3}{6}\} = 0.5$, which is substantial and will lead to incorrect merges between two different relationships (`IOC` and `ISO`).*

Using negative incompatibility, we can compute $w^-(B_1,B_3)$ as $-\max\{\frac{3}{6}, \frac{3}{6}\} = -0.5$, since the third, forth and sixth rows conflict between the two tables, and both tables have 6 rows. This suggests that B_1 and B_3 have substantial conflicts, indicating that a merge will be inappropriate.

In comparison, for B_1 in Table 2.8a and B_2 in Table 2.8b, which talk about the same relationship of `IOC`, their conflict set is empty and $w^-(B_1,B_2) = 0$, indicating that we do not have negative evidence to suggest that they are incompatible.

2.4.2 Problem Formulation for Synthesis

We use a graph $G = (\mathscr{B}, E)$ to model candidate tables and their relationships, where $\mathscr{B}$ is the union of all binary tables produced in the preprocessing step in Section 2.3. In G each vertex represents a table $B \in \mathscr{B}$. Furthermore, for each pairs of vertices $B, B' \in \mathscr{B}$, we use compatibility scores $w^+(B,B')$ and incompatibility scores $w^-(B,B')$ as the positive and negative edge weights of the graph.

Example 2.4.4. *Given the tables B_1, B_2 and B_3 in Table 2.8, we can represent them and their compatibility relationships as a graph as in Figure 2.3(a).*

As discussed in Example 2.4.2, the positive compatibility between $w^+(B_1,B_2) = 0.67$, which is shown as solid edge with positive weight in this graph. Similarly we have negative

edge weights like $w^-(B_1, B_3) = -0.5$ *as discussed in Example 2.4.3. This graph omits edges with a weight of 0, such as* $w^-(B_1, B_2)$.

Since we need to synthesize compatible tables into larger mapping relationships, in the context of graph G we need to group compatible vertices/tables together. This naturally corresponds to a partitioning $\mathscr{P} = \{P_1, P_2, ...\}$ of $\mathscr{B}$, where each $P_i \subseteq \mathscr{B}$ represents a subset of tables that can be synthesized into one relationship. Since different partitions correspond to distinct relationships, the partitioning should be disjoint ($P_i \cap P_j = \emptyset, i \neq j$), and they should collectively cover $\mathscr{B}$, or $\bigcup_{P \in \mathscr{P}} P = \mathscr{B}$.

Intuitively, there are many ways to partition $\mathscr{B}$ disjointly, but we want to find a good partitioning that has the following desirable properties: (1) compatible tables are grouped together as much as possible to improve coverage of individual mapping relationships; and (2) incompatible tables should not be placed in the same partition.

We translate these intuitive requirements into an optimization problem. First, we want each partition P to have as many compatible tables as possible. Let $w^+(P)$ be the sum of positive compatibility in a partition P:

$$w^+(P) = \sum_{B_i, B_j \in P, i<j} w^+(B_i, B_j)$$

We want to maximize the sum of this score across all partitions, or $\sum_{P \in \mathscr{P}} w^+(P)$. This is our optimization objective.

On the other hand, we do not want to put incompatible tables with non-trivial w^- scores, such as B_1 and B_3 in Example 2.4.4, in the same partition. Since we disallow this to happen, we treat edges with negative scores w^- below a threshold τ as *hard-constraints*. Note that a negative threshold τ (e.g., -0.2) is used in place of 0 because we do not over-penalize tables with slight inconsistency due to minor quality and extraction issues. We ignore the rest with insignificant negative scores by essentially forcing them to 0. Let $w^-(P)$ be the sum of substantial negative weights in P defined below.

$$w^-(P) = \sum_{B_i,B_j \in P, w^-(B_i,B_j) < \tau} w^-(B_i,B_j)$$

We use this as a constraint of our formulation – we want no edges in the same partition to have substantial conflicts, or, $w^-(P) = 0, \forall P \in \mathscr{P}$.

Putting these together, we formulate table synthesis as follows.

Problem 2.4.5 (Table Synthesis)**.**

$$max \qquad \sum_{P \in \mathscr{P}} w^+(P) \tag{2.5}$$

$$s.t. \qquad \sum_{P \in \mathscr{P}} w^-(P) = 0 \tag{2.6}$$

$$P_i \bigcap P_j = \emptyset, \qquad \forall P_i \neq P_j \tag{2.7}$$

$$\bigcup_{P \in \mathscr{P}} P = \mathscr{B} \tag{2.8}$$

By placing compatible tables in the same partition, we score more in the objective function in Equation (2.5), but at the same time Equation (2.6) guarantees that no conflicting negative edge can be in the same partition. Equation (2.7) and (2.8) are used to ensure that $\mathscr{P}$ is a proper disjoint partitioning.

Example 2.4.6. *We revisit the example in Figure 2.3(a). Using the formulation above, it can be verified that the best partitioning is* $\{\{B_1, B_2\}, \{B_3, B_4, B_5\}\}$*, which groups two ISO tables and three IOC tables into separate partitions. This partitioning has a total score of* 2.77 *based on Equation* (2.5)*, without violating constraints in Equation* (2.6) *by not placing negative edges in the same partition.*

It is worth noting that existing techniques like schema matching [132] only consider positive similarity (because FD do not generally hold in tables), and as a result merge all 5 tables in this example, producing results of low quality.

Theorem 2.4.7. *The problem Table-Synthesis is NP-hard.*

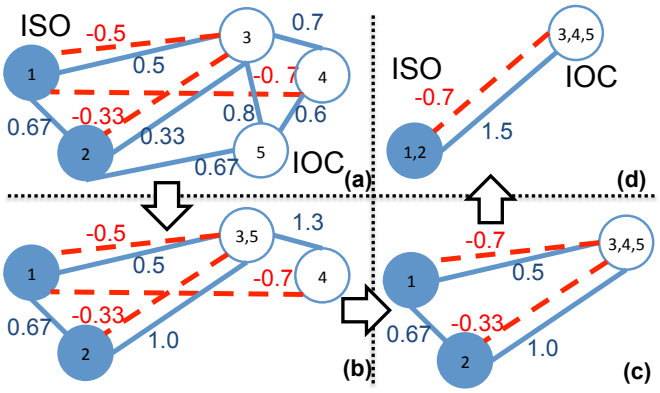


Figure 2.3: Graph representation of candidate tables. Solid vertices on the left represent tables for ISO codes; hollow vertices on the right represent tables for IOC codes. Furthermore, solid edges indicate positive compatibility, while dashed edges indicate negative incompatibility. Edges with weight of 0 are omitted on the graph.

We prove this using a reduction from graph multi-cut. There also exists a trichotomy of complexity depending on the number of negative edges in the graph.

Proof. We prove it by showing that Problem 2.4.5 is a more general case of a typical multi-cut problem in a weighted graph [77]. Given an undirected graph $G_C = (V_C, E_C)$, a weight function w_C of the edges, and a set of k_C pairs of distinct vertices (s_i, t_i), the multi-cut problem is to find the minimum weight set of edges of G_C that disconnect every s_i from t_i. The multi-cut problem is NP-hard [38].

Now we transform $G_C = (V_C, E_C)$ to graph $G = (\mathscr{B}, E)$ as follows: (i) We first divide the weights by a large number, $\max\{w_C(v_i, v_j)\}$, to change the range of weights to $(0, 1]$. (ii) We define $\mathscr{B} = V_C$ and $E = E_C$. (iii) We make positive weights $w^+(v_i, v_j) = w^+(v_j, v_i) = w_C(v_i, v_j)$. (iv) For each pair of vertices (s_i, t_i), we make negative weights $w^-(s_i, t_i) = w^-(t_i, s_i) = -1 < \tau$.

As a result, each partitioning $\mathscr{P}$ in Problem 2.4.5 corresponds to exactly one cut E_{cut} in the above multi-cut problem because: (i) Constraint (2.6) guarantees that s_i and t_i are never in the same partition. (ii) The edges across partitions are E_{cut} (i.e. the set of edges to be removed) in the multi-cut problem. (iii) Let $w^+(\mathscr{P})$ be the objective function $\sum_{P \in \mathscr{P}} w^+(P)$

of Problem 2.4.5, the sum of weights of the graph be $w_C(G_C)$, and the weight of cut be $w_C(E_{cut})$. Then $w^+(\mathscr{P}) + w_C(E_{cut}) = w_C(G_C)$. So maximizing $w^+(\mathscr{P})$ is equivalent to minimizing $w_C(E_{cut})$. Therefore we reduce the multi-cut problem to Problem 2.4.5.

So Problem 2.4.5 is NP-hard. □

What is interesting is that there exists a trichotomy result in terms of complexity [41]. Specifically, if the graph has exactly 1 negative edge, the problem is equivalent to min-cut, max-flow because we can make the pair of vertices incident to the negative edge as source and sink, respectively. When there exist 2 negative edges, the problem can be solved in polynomial time using results from [177]. In the more general case when there are no fewer than 3 negative edges, the problem becomes NP-hard.

Despite the hardness, there is a $O(\log N)$-approximation algorithm for the loss-minimization version of Problem 2.4.5. Specifically, the loss-minimization version of the problem can be written as follows, which minimizes the positive edge weights that are lost as a result of the partitioning that disconnects all the negative edges.

Problem 2.4.8 (Loss Minimization)**.**

$$min \quad \sum_{B\in P_i, B'\in P_j, i\neq j} w^+(B,B') \tag{2.9}$$

$$s.t. \quad \sum_{P\in\mathscr{P}} w^-(P) = 0 \tag{2.10}$$

$$P_i \bigcap P_j = \emptyset, \qquad \forall P_i \neq P_j \tag{2.11}$$

$$\bigcup_{P\in\mathscr{P}} P = \mathscr{B} \tag{2.12}$$

Using standard embedding techniques, we can encode partition decisions using distance variables d_{ij}. $d_{ij} = 0$ if vertices B_i and B_j are in the same partition, and $d_{ij} = 1$ if they are in different partitions. This produces the following formulation.

Problem 2.4.9 (Embedding)**.**

$$
\begin{aligned}
\textit{min} \quad & \sum w^{+}(B_i, B_j) \cdot d_{ij} \\
\textit{s.t.} \quad & d_{ij} + d_{jk} \geq d_{ik}, \quad \forall i, j, k && (2.13) \\
& d_{ij} = d_{ji} && (2.14) \\
& d_{ij} \in \{0, 1\} && (2.15) \\
& d_{ij} = 1, \qquad \forall w^{-}(B_i, B_j) < \tau && (2.16)
\end{aligned}
$$

This problem is known to be APX-hard [38]. We can relax it by replacing the integrality constraint with $d_{ij} \in [0,1]$ to make it an LP, which can then be solved using a standard solver in polynomial time.

Using the optimal fractional solution from the LP, one can round such a solution in a region-growing procedure [10, 59, 156] that finds an integral solution close to the fractional solution. This randomized rounding process guarantees $O(\log N)$ approximation for the loss minimization version of the problem.

Such an approximation scheme requires to model each pair of vertices as a decision variable d_{ij}, and then solve the associated LP before applying randomized rounding. While it may be practical for problems of moderate sizes, we are dealing with graphs with millions of vertices, where solving an LP with a quadratic number of variables is clearly infeasible.

As a result, we use an efficient heuristic to perform greedy synthesis. Specifically, we initially treat each vertex as a partition. We then iteratively merge a pair of partitions (P_1, P_2) that are the most compatible to get a new partition P', and update the remaining positive/negative edges. The algorithm terminates when no partitions can be merged. Algorithm 3 shows the pseudo-code for table synthesis.

Efficiency. While the procedure above appears straightforward for graphs that fit in a single machine, scaling to large graphs on Map-Reduce is not straightforward. We use a

Algorithm 3: Table-Synthesis by Partitioning

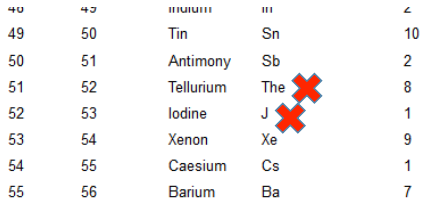
Input: Graph $G = (\mathscr{B}, E)$, threshold τ
Output: Set of Partitions $\mathscr{P}$

$P(B_i) \Leftarrow \{B_i\}, \ \forall B_i \in \mathscr{B}$
$\mathscr{B}_P \Leftarrow \bigcup_{B_i \in \mathscr{B}} \{P(B_i)\}$
$E_P \Leftarrow \bigcup_{(B_i,B_j) \in E} \{(P(B_i), P(B_j))\}$
$w_P^+(P(B_i), P(B_j)) \Leftarrow w^+(B_i, B_j)$
$w_P^-(P(B_i), P(B_j)) \Leftarrow w^-(B_i, B_j)$
$G_P \Leftarrow (\mathscr{B}_P, E_P)$
while *true* **do**
    $e(P_1, P_2) \Leftarrow \underset{P_1 \neq P_2, w_P^-(P_1,P_2) \geq \tau}{arg\,max} (w_P^+(P_1, P_2))$
    if e =*NULL* **then**
        break
    $P' \Leftarrow P_1 \cup P_2$
    Add P' and related edges into $\mathscr{B}_P$ and E_P
    foreach $P_i \notin \{P_1, P_2\}$ **do**
        $w_P^+(P_i, P') \Leftarrow w_P^+(P', P_i) \Leftarrow w_P^+(P_i, P_1) + w_P^+(P_i, P_2)$
        $w_P^-(P_i, P') \Leftarrow w_P^-(P', P_i) \Leftarrow \min\{w_P^-(P_i, P_1), w_P^-(P_i, P_2)\}$
    Remove P_1, P_2 and related edges from $\mathscr{B}_P$ and E_P
$\mathscr{P} \Leftarrow \mathscr{B}_P$

divide-and-conquer approach to first produce components that are connected non-trivially by positive edges on the full graph, and then look at each subgraph individually.

We use the Hash-to-Min algorithm to compute connected components on Map-Reduce [32]. This algorithm treats every vertex and its neighbors as a cluster initially. Then for each cluster, it sends a message of the cluster ID to all its members. Next every vertex chooses the minimum cluster ID it receives and propagate this minimum ID as the new ID of all the other clusters who sends message to it. The algorithm iteratively apply the above steps until convergence. This algorithm solves our problem very efficiently.

Now given a subgraph, we apply Algorithm 3 to solve Problem 2.4.5. Set union and lookup are two frequent operations in Algorithm 3. So we use a disjoint-set data structure to speed up the process [76]. Its idea is to maintain a tree to represent each set so that union and lookup of the tree root are much faster than a naïve set operation.



49	50	Tin	Sn	10
50	51	Antimony	Sb	2
51	52	Tellurium	The	8
52	53	Iodine	J	1
53	54	Xenon	Xe	9
54	55	Caesium	Cs	1
55	56	Barium	Ba	7

Figure 2.4: A real table with errors that can cause conflicts.

Example 2.4.10. *Figure 2.3 shows how Algorithm 3 works on a small graph. The algorithm first merges* $\{B_3\}$ *and* $\{B_5\}$ *to get Figure 2.3b because Edge* $(\{B_3\},\{B_5\})$ *has the greatest weight. The weight of Edge* $(\{B_2\},\{B_3,B_5\})$ *changes as* $w^+(\{B_2\},\{B_3,B_5\}) \Leftarrow w^+(\{B_2\},\{B_3\}) + w^+(\{B_2\},\{B_5\})$. *The weight of Edge* $(\{B_4\},\{B_3,B_5\})$ *also changes similarly.*

The algorithm then merges $\{B_3,B_5\}$ *and* $\{B_4\}$ *to get Figure 2.3c and finally combines* $\{B_1\}$ *and* $\{B_2\}$ *to get Figure 2.3d. The algorithm stops because of the negative weight between* $\{B_1,B_2\}$ *and* $\{B_3,B_4,B_5\}$.

2.4.2.1 Conflict Resolution

We observe that synthesized relations often have conflicts that require post-processing. Specifically, when we union all tables in the same partition together, there will be a small fraction of rows that share the same left-hand-side value, but have different right-hand-side values. This could be due to quality issues in the original input tables, such as the example in Figure 2.4 that has incorrect chemical symbols for two of the rows (the symbol of `Tellurium` should be `Te` and `Tellurium` should be `I`). Quality issues like this are actually common in large corpus, and manifest themselves as inconsistent mappings in synthesized results. Since the majority of tables in the partition should agree with the ground-truth mapping, we resolve conflicts by removing the least number of low-quality tables, such that the resulting partition has no conflicts.

Let P be a partition with candidate tables $\{B_1,B_2,...\}$, each of which is a set of value pairs $B_i = \{(l,r)\}$. Recall that in Section 2.4.1 we define a *conflict set* $F(B,B')$ to be

$\{l|(l,r) \in B, (l,r') \in B', r \neq r'\}$. We can again leverage synonyms and do not treat $(l,r),(l,r')$ as conflicts if (r,r') are known to be synonyms.

Now we want to find out the largest subset $P_T \subseteq \mathscr{P}$ such that no two tables in P_T conflict with each other, which can be formulated as follows.

Problem 2.4.11 (Conflict Resolution)**.**

$$\begin{aligned} max \quad & \left| \bigcup_{B_i \in P_T} B_i \right| \\ s.t. \quad & F(B_i, B_j) = \emptyset, \quad \forall B_i, B_j \in P_T \end{aligned} \tag{2.17}$$

The objective is to include as many value pairs as possible, under the constraint that no pairs of tables in the selected subset P_T can have conflict. This problem is NP-hard (reduction from Independent Set).

Proof. We prove the hardness of conflict resolution by reducing the maximum independent set (MIS) problem to it. Given a graph $G_M = (V_M, E_M)$ in MIS, we correspondingly build a partition $P = \{B_1, B_2, ...\}$ in Problem 2.4.11 as follows: (1) For each $v_m \in V_M$, we create a B_m. (2) For each $e_m(v_i, v_j) \in E_M$, we create a pair of contradicting value pairs $((l,r),(l,r'))$. We add (l,r) to B_i and (l,r') to B_j. (3) Let the maximum vertex degree of G_M be deg. We add dummy value pairs to each B_i to make $|B_i| = deg$. These dummy value pairs do not conflict with any existing value pairs. Obviously, the MIS problem has a solution with size S_{MIS}, if and only if Problem 2.4.11 has a solution with weight $S_{MIS} \cdot deg$. □

Since this problem is NP-hard, we iteratively find and remove a value pair that conflicts with the most other value pairs. Algorithm 4 iteratively finds value pairs that conflict with the most other value pairs and removes its candidate table. Specifically, given a value pair (v_1, v_2), Line 3 to Line 5 counts the number of conflicting value pairs. Line 6 to Line 9 finds the candidate that introduces the most conflicts and removes it. In practice, we maintain an

index for each value pair and each candidate to keep track the number of conflicts. We use a heap that supports update to select the most conflicting candidate efficiently.

Algorithm 4: Conflict Resolution

Input: Partition $P = \{B_1, B_2, ...\}$
Output: P_T without conflict

$P_T \Leftarrow P$
while $\exists B_i, B_j \in P_T, |F(B_i, B_j)| > 0$ **do**
  $InstSet \Leftarrow \bigcup_{B_i \in P_T} B_i$
  foreach $(v_1, v_2) \in InstSet$ **do**
    $cnt_V(v_1, v_2) \Leftarrow$ # conflicting value pairs in $InstSet$
  foreach $B_i \in P_T$ **do**
    $cnt_B(B_i) \Leftarrow \max_{(v_1,v_2) \in B_i}\{cnt_V(v_1, v_2)\}$
  $B_i \Leftarrow arg\,max_{B_i \in P_T} cnt_B(B_i)$
  $P_T \Leftarrow P_T n\{B_i\}$

2.4.2.2 Table Expansion

Another potential issue is that for large mapping relationships such as (`airport-name`, `airport-code`) that has more than 10K instances, synthesized tables may still miss values that are unpopular with little or no presence in web tables. We see that synthesized relationships provide a robust "core", which can be used to bring in additional instances. We perform an optional expansion step, by using external data resources such as data.gov or spreadsheet files (*.xlsx*) crawled from other trustable web sources, that are more likely to be comprehensive (web tables on the other hand are often for human consumption and tend to be short). We compute the similarity and dissimilarity between our synthesized "cores" and these external sources, and merge if certain requirements are met. Note that this step can also happen at curation time with human users in the loop.

We compare the f-score before and after table expansion. Overall the effect is limited. F-score is improved substantially for only two cases, namely (`airport-name` $\rightarrow$ `IATA-code`) and (`airport-name` $\rightarrow$ `ICAO-code`). These two have over 10k instances in ground truth. So synthesis alone is not sufficient to recover the full relationship, and expansion brings more pronounced effect.

We note that there are many existing methods for conflict resolution [103] that can conceptually be applied to the post-processing step, and it is interesting to explore their applicability. Because we do not consider this post-processing step to be our key contribution, and we include this step here for completeness, we do not perform an exhaustive comparison.

2.4.3 Synthesized Mappings for Curation

While the synthesized mappings produced by our algorithm are generally of high quality, for many applications a very high precision is required. For example, for commercial spreadsheet software like Excel, any error introduced by black-box algorithms can be hard to detect by users, but has damaging consequences and thus unacceptable. In such settings, our approach of pre-computing all candidate mappings from table corpora allows humans to inspect and curate these mappings to ensure very high accuracy. High-quality mappings produced by automatic algorithms can greatly reduce the effort required by human curators.

It is interesting to note that synthesized results we produce have a natural notion of importance/popularity. Specifically, for each synthesized mapping, we have statistics such as the number of web domains whose tables contributed to this mapping, and how many raw tables are synthesized in the same cluster, etc. Such statistics are very well correlated to the importance of the mapping, because the more it occurs in the table corpus, the more likely it is frequently used and important. This property makes results produced by our approach amenable to human curation – instead of looking at a full corpus with millions of tables, one just needs to look at synthesized results popular enough.

In our experiments using a web corpus, we only use about 60K synthesized mappings from at least 8 independent web domains, which is orders of magnitude less than the the number of input tables. Additional filtering can be performed to further prune out numeric and temporal relationships.

While most mappings are static that rarely change, some are temporal in nature and may be changing over time. But like knowledge-bases used by search engines that also face the same "data freshness" problem (e.g., when a famous actor gets newly married, the knowledge-card used by search engines should reflect that new fact within a short period of time), algorithms and human curation can for the most part mitigate this problem (e.g., regularly refreshing the data by rerunning the pipeline and alert human curator for changes). Additional mechanisms include crowd-sourcing that allows users to report/flag stale values for them to be corrected. We would like to note that because a large fraction of the mappings harvested are static in nature, a one-shot curation of just these mappings can already produce significant values to a variety of applications.

2.5 Experiments

2.5.1 Experimental Setup

2.5.1.1 Table Corpus

We use two table corpora for our evaluation.

The first table corpus, henceforth denoted as `Web`, has over 100 million tables crawled and extracted from the public web. These tables cover diverse domains of interests.

The second table corpus, denoted as `Enterprise`, has about 500K tables extracted from spreadsheets files crawled from the intranet of a large IT company.

2.5.1.2 Computing Environment

We implemented algorithms described in this chapter as Map-Reduce programs. We ran our jobs in a large Map-Reduce cluster, alongside with other production jobs. Our input for `Web` has about 223M two-column tables with a size of over 200GB.

list of countries and capitals	list of pokemons and categories
list of car models and makes	list of amino acids and symbols

Figure 2.5: Example queries with "list of A and B"

FIPS 5-2	ISO 3166-1 Alpha-3
FIPS 10-4	ISO 3166-1 Numeric
IANA Country Code	ITU-R Country Code
IATA Airport Code	ITU-T Country Calling Code
ICAO Airport Code	MARC Country Code
IOC Country Code	NUTS (EU)
ISO 3166-1 Alpha-2	SGC Codes (Canada)

Figure 2.6: Geocoding Systems

2.5.1.3 Benchmarks

We have built a benchmark dataset to evaluate our framework on `Web` table corpus [2]. This benchmark dataset contains 80 desirable mapping relationships that we manually curated. These relationships are collected from two sources.

• Geocoding: We observe that geography is a common domain with rich mapping relationships that are often used in auto-join and auto-correction scenarios. Examples here include geographical and administrative coding such as country code, state code, etc. So we take 14 cases from a Wikipedia list of geocoding systems[3]. We omit codes that are impossible to enumerate such as military grid reference system, and ones not completely listed on Wikipedia such as HASC code. Figure 2.6 lists all cases we take.

• Query Log: We sample queries of the pattern "list of A and B" in Bing query logs that search for mapping relationships. Figure 2.5 shows a few examples with true mappings.

For `Web`, after selecting mapping relationships, we curate instances for each relationship, by combining data collected from web tables as well as knowledge bases. Specifically,

[2]Mappings in the web benchmark is available at `https://www.microsoft.com/en-us/research/publication/synthesizing-mapping-relationships-using-table-corpus/`

[3]`https://en.wikipedia.org/wiki/Geocoding`

we find a group of tables for each relationship, and then manually select high-quality ones to merge into the ground truth. Finally we combine these high-quality web tables with instances in Freebase and YAGO if they have coverage. Note that the resulting mapping relationships have rich synonyms for the same entity (e.g., as shown in Table 2.6), as well as more comprehensive coverage for instances. Constructing such a benchmark set, and ensuring its correctness/completeness is a time-consuming process. We intend to publish this benchmark set online to facilitate future research in this area.

`Enterprise` is more difficult to benchmark because of the difficulty in ensuring completeness of instances in certain mappings – the ground truth may be in master databases for which we have no access (e.g., `employee` and `login-alias`). Nevertheless, we built 30 best effort benchmark cases. Recall results on these tests should be interpreted as relative-recall given the difficulty to ensure completeness.

2.5.1.4 Metrics

We use the standard precision, recall and f-score to measure the performance. Let $B^* = \{(l^*, r^*)\}$ be a ground truth mapping, and $B = \{(l, r)\}$ be a synthesized relationship for which we want to evaluate its quality. The precision of B is defined as $\frac{|B \cap B^*|}{|B|}$, the recall is $\frac{|B \cap B^*|}{|B^*|}$, and the f-score is $\frac{2\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$.

2.5.1.5 Methods Compared

We compare the following methods.

• `UnionDomain`. Ling and Halevy et al. [107] propose to union together tables within the same website domain, if their column names are identical but row values are disjoint. We apply this technique by essentially grouping tables based on column names and domain names. We evaluate the resulting union tables against each benchmark case by picking the union table with the highest F-score.

• `UnionWeb`. Noticing that only union-ing tables in the same domain may be restrictive and missing instances for large relationships, we extend the previous approach to also merge

tables with the same column names across the web, and evaluate all benchmark cases like above. This is a variant of `UnionDomain`.

• `Synthesis`. This is our approach that synthesizes mapping relationships as described in Section 2.4.

• `SynthesisPos`. This is the same as `Synthesis` except that it does not use the negative signals induced by FDs. This helps us to understand the usefulness of negative signals.

• `WiseIntegrator [69, 70]`. This is a representative method in a notable branch in schema matching that collectively matches schemas extracted from Web forms. It measures the similarity between candidates using linguistic analysis of attribute names and value types, etc., and performs a greedy clustering to group similar attributes.

• `SchemaCC`. In this method, we mimic pair-wise schema matchers that use the same positive/negative similarity as our approach. Because match decisions are pair-wise, we aggregate these to a group-level based on transitivity (e.g., if table A matches B and B matches C, then A also matches C). This is implemented as connected components on very large graphs, where edges are threshold based on a weighted combination of positive/negative scores. We tested different thresholds in the range of $[0,1]$ and report the best result.

• `SchemaPosCC`. This is the same as `SchemaCC` but without negative signals induced by FDs, since they are not explored in the schema matching literature. We again test thresholds in $[0,1]$ and report the best number.

• `Correlation [31]`. In this method, we again mimic pair-wise schema matchers with the same positive/negative scores as `Synthesis`. Instead of using connected components for aggregation as in `SchemaCC` above, here we instead use the correlation clustering that handles graphs with both positive or negative weights. We implement the state-of-the-art correlation clusterin on map-reduce [31], which requires $O(\log|V|\cdot\Delta^+)$ iterations and takes a long time to converge ($|V|$ is the number of vertices of the graph and Δ^+ is the maximum degree of all vertices). We timeout after 20 hours and evaluate the results at that point.

• WikiTable. Wikipedia has many high-quality tables covering various domains, many of which have mapping relationships. To understand the quality of using raw tables instead of performing synthesis, we also evaluate each benchmark case by finding best pair of columns in a Wikipedia table that has the highest in F-score.

• WebTable. This method is very similar to the previous WikiTable, but use all tables in the Web corpus instead of just Wikipedia ones.

• Freebase. Freebase [17] is a well-known knowledge base that has been widely used. We obtained its RDF dump[4] and extract relationships by grouping RDF triples by their predicates. We treat the subject $\rightarrow$ object as one candidate relationship, and the object $\rightarrow$ subject as another candidate.

• YAGO. YAGO [146] is another public knowledge base that is extensively used. We process a YAGO data dump similar to Freebase, by grouping YAGO RDF triples using their predicates to form subject-object and object-subject relationships.

Note that in all these cases, we score each benchmark case by picking the relationship in each data set that has the best f-score. This is favorable to all the methods – a human who wishes to pick the best relationship to be used as mappings, and who could afford to inspect all these tables, would effectively pick the same tables.

2.5.2 Quality Comparison

Figure 2.7 shows the average f-score, precision and recall across all 80 benchmark cases in the Web benchmark for all methods compared. Synthesis scores the best in average recall (0.88) and f-score (0.90), while WikiTable has the best average precision (0.98) [5].

In comparison, using only raw tables from WikiTable with no synthesis has high precision but low recall, because not only are certain instances and synonyms missing (these

[4] https://developers.google.com/freebase/

[5] Since WikiTable methods miss many relationships, we exclude cases whose precision is close to 0 from the average-precision computation. This makes the average precision favorable to WikiTable. The same is applied to other table and knowledge based methods.

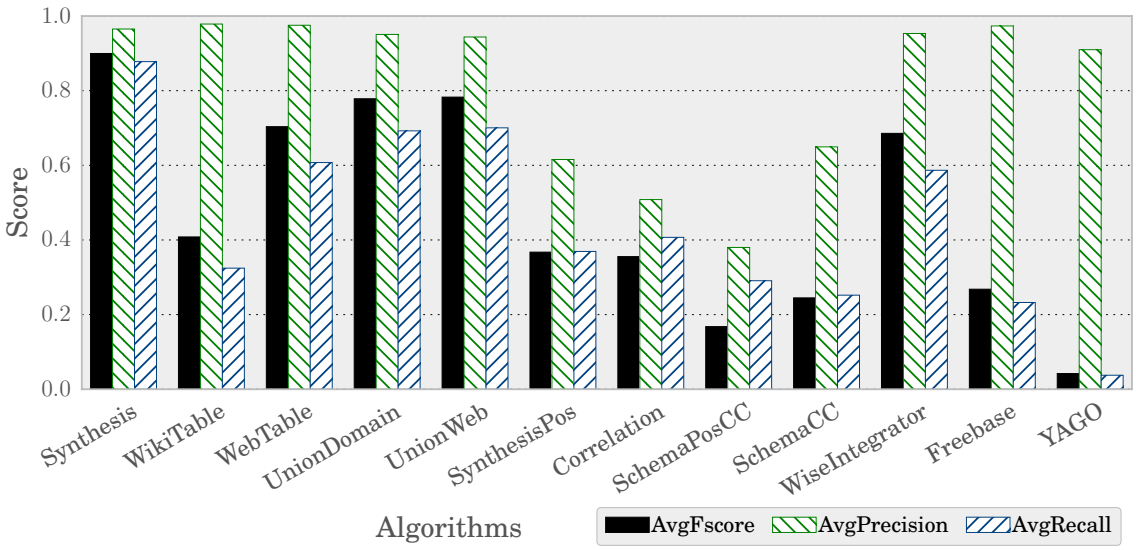


Figure 2.7: Average f-score, precision and recall comparison.

tables tend to be short for human consumption), many relationships are also missing altogether from `WikiTable`. So the approach of manually going over high-quality `WikiTable` to curate mapping relationships is unlikely to be sufficient.

The `WebTable` approach uses raw tables similar to `WikiTable`, but considers tables not limited to the Wikipedia domain and thus has substantially better recall. While the precision of `WebTable` and `Synthesis` are comparable, the recall of `Synthesis` is substantially higher (0.88 vs. 0.32). Despite this, we want to note that the setup of this comparison of is very favorable for `WebTable` – we select the best table among the hundreds of millions of raw tables in `WebTable`, whereas in `Synthesis` we only use relations synthesized from over 8 website domains that is three orders of magnitude less (Section 2.4.3). Because it is not possible for human to go over millions of tables to pick useful mappings in practice, `WebTable` only provides an upper-bound of what can be achieved and not really a realistic solution.

`UnionDomain` and `UnionWeb` synthesize tables based on table column names and domain names. The recall of these two approaches is considerably better than `WikiTable` and `WebTable`, showing the benefit of performing table synthesis. However, this group of

approaches merge tables only based on column names, which are known to be uninformative and undescriptive in many cases. We observe that when applied to the whole web, this often leads to over-grouping and under-grouping. The overall f-scores of these approaches are the best among all existing methods, but still lag behind `Synthesis`, which uses values that are more indicative of table compatibility.

`SynthesisPos` uses the same algorithm as `Synthesis` but does not consider the negative incompatibility induced by FDs. It is interesting to observe that result quality suffers substantially, which underlines the importance of the negative signals.

`SchemaCC` performs substantially worse than `Synthesis`. Recall that it uses the same positive/negative signals, but aggregate pair-wise match decisions using connected components. This simple aggregation tends to over-group and under-group different tables, producing undesirable table clusters.

`SchemaPosCC` ignores the negative signals used in `SchemaCC`, since FD-induced negative signals are not explored in schema matching. Unsurprisingly, result quality drops even further.

`Correlation` is similar to `SchemaCC` that also mimics schema matchers with same signals, but aggregate using correlation clustering. Overall, its f-score is better than `SchemaCC`, but is still worse than `Synthesis`. We think there are two main reasons why it does not work well. First, at the conceptual level, the objective of correlation clustering is the sum of positive and negative edges. Because the number of table pairs that would be in different clusters far exceeds the ones that should be in the same clusters, making negative edges dominate the objective function. However, in our problem, we should actually only care about whether tables in the same clusters correspond to the identical mapping, which are the intra-cluster positive edges that are more precisely modeled in our objective function. Second, a shortcoming of the parallel-pivot algorithm [31] is that it only looks at a small neighborhood for clusters (i.e. one-hop neighbors of cluster centers) for efficiency. When small tables in the same mapping form a chain of connected components, looking at the im-

mediate neighborhood of a pivot (cluster center) will misses most other tables, producing results with low recall.

We implemented the collective schema-matching method `WiseIntegrator`. It performs reasonably well but still lags behind `Synthesis`, mainly because of the difference in how scores are aggregated to produce holistic matches.

2.5.2.1 Detailed Example Mappings

Now we discuss additional synthesized mappings that are not in the benchmark. Figure 2.8 lists additional popular mappings synthesized using `Web`. Many relationships involve geographic information such as (`US-city` → `state-abbreviation`), (`India-railway-station` → `state`), (`UK-county` → `country`), etc. There are a variety of other relationships such as (`wind` → `Beaufort-scale`), (`ASCII-abbre-viation` → `code`), (`automobile` → `type`) etc. We find reasonable meanings of these binary relationships and consider them to be high quality.

However, certain synthesized binary relationships are less ideal as mappings. We show such cases in Figure 2.9. For example, certain relationships are temporal that only hold for a period of time. Examples like (`F1-driver`, `team`), (`English-football-club`, `points`), (`college-football-team`, `ranking`) are in this category. Because this leads to many mappings of the same type that are true in different point of time (e.g., points of soccer teams), additional reasoning of conflicts between synthesized clusters can potentially identify such temporal mappings. We leave improving results in this regard as future work.

Certain tables are used repeatedly for formatting purpose, whose values would get extracted as popular mappings. For example, the (`month`, `month`) in Figure 2.9 maps `January` to `July`, `Feb` to `August` and so on, simply because many pages list 12 month calendar as two column tables. Results in this category are not significant in numbers, and should be relatively easy for human to prune out.

Mapping Relationship	Example Instances	Mapping Relationship	Example Instances
(US-city, state-abbr.)	(New York, NY), (Chicago, IL), ...	(movie, year)	(Pulp Fiction, 1994), (Forrest Gump, 1994), ...
(gun-powder-name, company)	(Varget, Hodgdon), (RL-15, Alliant), ...	(movie, distributor)	(The Dark Knight Rises, WB), (Life of Pi, Fox), ...
(UK-county, country)	(Suffolk, England), (Lothian, Scotland), ...	(ODBC-configuration, default-value)	(odbc.check_persistent, on), (odbc.default_db, no value), ...
(India-railway-station, state)	(Vadodara Junction, Gujarat), (Itarsi Junction, Madhya Pradesh), ...	(automobile, type)	(F-150, truck), (Escape, SUV), ...
(wind, Beaufort-scale)	(gentle breeze, 3), (storm, 10), ...	(family-member, gender)	(Mother, F), (Brother, M), ...
(state/province abbr., country)	(QLD, AU), (ON, CA), ...	(ASCII-abbr., code)	(NUL, 0), (ACK, 6), ...
(ISO3166-1-Alpha-3, ISO3166-1-Alpha-2)	(USA, US), (FRA, FR), ...	(ISO-4217-currency- code, num)	(USD, 840), (EUR, 978), ...

Figure 2.8: Additional mappings synthesized from Web.

Mapping Relationship	Example Instances	Mapping Relationship	Example Instances
(MiLB-leagues, level)	(PCL, AAA), (IL, AAA), ...	(college-football-team, ranking)	(Alabama, 1), (Clemson, 3), ...
(baseball-team, league)	(NYY, AL), (LAD, NL), ...	(college-football-team, score)	(Stanford, 5-0), (Michigan, 5-0), ...
(English-football-club, points)	(Manchester City, 16), (Liverpool, 17), ...	(football-player, team)	(Marques Colston, NO), (Victor Cruz, NYG), ...
(US-soccer-club, points)	(Houston Dynamo, 48), (Chicago Fire, 49), ...	(month, month)	(January, July), (February, August), ...
(F1-driver, team)	(Sebastian Vettel, Ferrari), (Lewis Hamilton, Mercedes), ...	(day, hour)	(Monday, 7:30AM - 5:30PM), (Tuesday, 7:30AM - 5:30PM), ...

Figure 2.9: Synthesized relationships not ideal as mappings.

2.5.2.2 Usefulness of Mappings

We sample the top clusters produced based on popularity (the number of tables/domains contributing to the cluster). We classify the mapping corresponding to each cluster into three categories: Meaningful mapping (static), Meaningful mapping (temporal), and Meaningless mapping. For top 500 clusters we inspected, 49.6% are static, 37.8% are temporal, and only 12.6% are meaningless, which is encouraging.

2.5.2.3 Individual Cases

Methods using knowledge bases Freebase and YAGO have reasonable precision, which is expected because they are extensively curated. The recall numbers of these methods, however, are substantially lower, because a significant fraction of useful mappings are

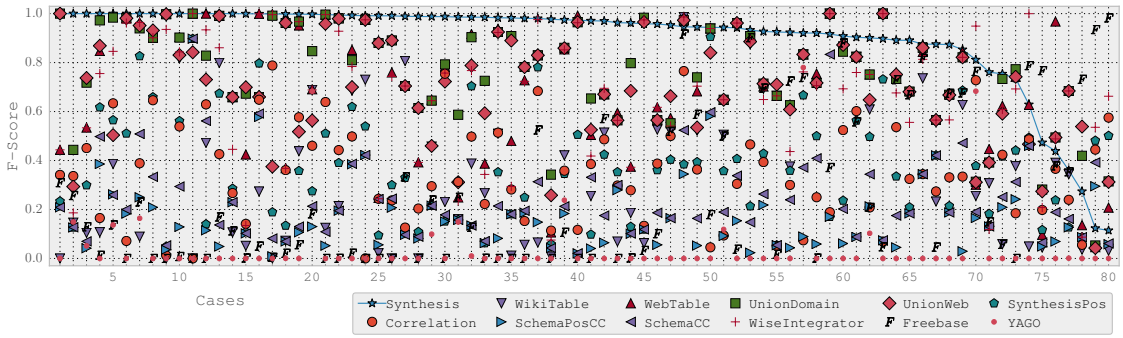


Figure 2.10: Comparison with alternatives on individual cases (Sorted by f-score of our Synthesis approach).

missing from existing knowledge bases. This indicates that knowledge bases alone are unlikely to be insufficient for harvesting rich mappings.

Figure 2.10 gives detailed quality numbers for individual cases in the benchmark. The overall observation here is consistent with Figure 2.7. We can see that for a large fraction of test cases, `Synthesis` produces results of high quality, which are amenable to further human curation before they are applied in data-driven applications. It is interesting to note that even when high-precision methods `WikiTable` and `Freebase` already have a complete table covering instances in certain benchmark cases such as `chemical element` and `country code`, their f-scores are still low despite almost perfect precision. This is because the ideal ground truth mapping should contain many synonymous names for the same entity (e.g., shown in Figure 2.6 for `country code`). In fact, the results `Synthesis` produces have over 470 entries for `country code` (compared to around 200 distinct countries), and over 200 entries for `chemical element` (compared to about 100 distinct ones). Methods like `WikiTable` and `Freebase` tend to have only one name mention for the same entity in one table, thus producing inferior scores for recall. As we have discussed in the introduction, such synonymous entity names are important for applications like auto-join and auto-correct, since a user data table can always have one name but not the other.

Interestingly, for a number of cases where `Synthesis` does not produce satisfactory results (towards the right of the figure), `Freebase` performs surprisingly well. It appears

that for domains like chemicals, mappings such as (Case 80: `chemical-compound` $\rightarrow$ `formula`) and (Case 74: `substance` $\rightarrow$ `CAS number`) have little web presence, which gives limited scope for synthesis using tables. On the other hand, `Freebase` has many structured data sets curated by human from specialized data sources covering different domains, thus providing better coverage where no techniques using web tables gives reasonable performance. We believe this shows that knowledge bases are valuable as a source for mapping tables, which in fact can be complementary to `Synthesis` for producing mapping relationships.

An issue we notice for `Synthesis`, is that while it already distills millions of raw tables into popular relations that requires considerable less human efforts to curate, in some cases it still produces many somewhat redundant clusters for the same relationship because inconsistency in value representations often lead to incompatible clusters that cannot be merged. Optimizing redundancy to further reduce human efforts is a useful area for future research.

2.5.3 Run-time Comparison

We analyze the the complexity of our approach in this section. The basic input of our problem is a graph $G = (V, E)$ where V represents candidate tables and E represents their similarity. The most expensive part of our algorithm is in table synthesis (Step 2) that computes edge similarity and performs iterative grouping.

Figure 2.11 compares the runtime of all approaches. Knowledge bases are the most efficient because it amounts to a lookup of the relation with the highest f-score among all relations. `WikiTable`, `WebTable`, `UnionDomain`, `UnionWeb`, and `WiseIntegrator` are all relatively efficient but requires scans of large table corpus. Our approach `Synthesis` usually finishes within 10 hours (e.g., using parameters suggested here). `Correlation` is clearly the slowest, as correlation clustering converges very slowly even using the state-of-the-art parallel implementation on map-reduce [31].

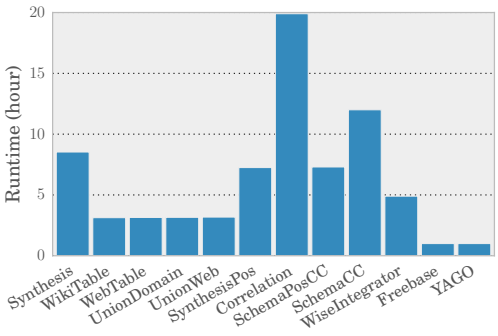


Figure 2.11: Runtime.

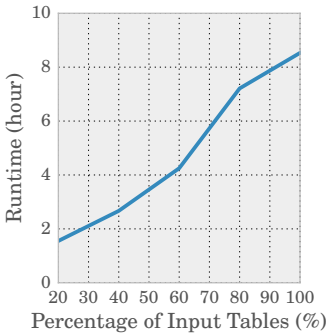


Figure 2.12: Scalability.

To test scalability of the proposed method, we sample $\{20\%, 40\%, 60\%, 80\%\}$ of the input data and measures execution time, as shown in Figure 2.12. The complexity of the algorithm depends on the number of edges $|E|$. In the worst case $|E|$ can be quadratic to the number of tables $|V|$, but in practice $|E|$ is usually almost linear to $|V|$ due to edge sparsity. Figure 2.12 suggests that the algorithm scales close to linearly to the input data, which is encouraging as it should also scale to even larger data sets with billions of tables.

2.5.4 Sensitivity Analysis

We analyze the effect of parameters used in `Synthesis`.

• θ. We use θ as a parameter when defining approximate mapping relationship, which is empirically set as 95%. When we vary θ between 93% and 97%, the number of resulting mappings change very little (by up to 1%). We have also reverse-engineered by calculating the degree of approximation in desirable ground truth mappings (e.g. `Springfield` $\rightarrow$ `Illinois` and `Springfield` $\rightarrow$ `Texas` will create a violation). 95% is sufficient to ensure that desired mappings will not be pruned incorrectly in almost all cases.

• τ. This parameter controls when we determine two candidates conflict. Our results suggest that the quality is generally insensitive to small τ. The performance peaks at around -0.05. In our other experiments we actually used $\tau = -0.2$ that also produces good quality.

	Synthesis	EntTable
Avg. (F-score, Prc., Rcl.)	(0.96, 0.96, 0.97)	(0.84, 0.99, 0.79)

Figure 2.13: Comparison with the alternative on Enterprise

• $\theta_{overlap}$ is a parameter for efficiency that determines the number of pruned edges $|E|$ in our graph. As $\theta_{overlap}$ increases, $|E|$ drops quickly. The quality of resulting clusters are insensitive to $\theta_{overlap}$.

• θ_{edge}. We make θ_{edge} the threshold to filter out edges with insignificant positive weight. Our experiment suggests that $\theta_{edge} = 0.85$ has the best performance.

2.5.5 Experiments on Enterprise

As we discussed earlier, unlike the Web domain where a large fraction of ground truth mappings can be constructed using common sense knowledge and online data sources, the ground truth mappings in the Enterprise domain is difficult to build. We are not familiar with many enterprise-specific data values and encodings in this corpus, which makes ensuring completeness and correctness of these mappings difficult.

We build 30 benchmark cases with best effort to ensure completeness (for some mappings the ground truth may be in master databases we have no access to). To put the quality numbers in perspective, we compare Synthesis with single-table based EntTable, which is similar to WebTable in Web. As Figure 2.13 suggests that Synthesis achieve significantly higher recall by merging small tables. Its precision is also high by avoiding merging conflicting content.

Figure 2.14 shows some examples of mapping relationships produced. A large fraction of relationships are indeed important mappings, such as (product-family $\rightarrow$ code), (profit-center $\rightarrow$ code), (data-center $\rightarrow$ region), etc. Most of these results are well-structured and look consistent (shown in the right column of Figure 2.14), which is a good indication that results produced are of high quality.

Mapping Relationship	Example Instances
(product-family, code)	(Access, ACCES), (Consumer Productivity, CORPO), ...
(profit-center, code)	(P10018, EQ-RU - Partner Support), (P10021, EQ-NA - PFE CPM), ...
(industry, vertical)	(Accommodation, Hospitality), (Accounting, Professional Services), ...
(ATU, country)	(Australia.01.EPG, Australia), (Australia.02.Commercial, Australia), ...
(data-center, region)	(Singapore IDC, APAC), (Dublin IDC3, EMEA), ...

Figure 2.14: Example mapping relationships and values, from the enterprise spreadsheets corpus

Just like in the `Web` domain, applications equipped with these mapping relationships and some human curation can perform intelligent operations such as auto-join as discussed earlier. We note that these mappings are specific to this enterprise in question. Using tables to build such relationships would be the only reasonable choice, since alternatives like knowledge bases would not exist in enterprise domains.

Inspecting the results produced in `Enterprise` does reveal interesting issues. For example, we observe that for certain mapping relationships, the results are of low quality with mixed data values and meta-data values (e.g., column headers). It turns out that in spreadsheets, tables with complex structures such as pivot tables are popular. These complex tables are usually not flat relational tables that create difficulty for correct extraction.

Overall, given that rich mappings are produced for a completely different `Enterprise` corpus, we believe that this exercise shows the promise of the `Synthesis` approach to generalize and produce mappings by just using a corpus of tables as input.

2.5.6 Effect of Conflict Resolution

Conflict resolution improves the f-score for 48 out of the 80 cases tested. On average, the precision increases from 0.903 to 0.965, while the average recall only dips slightly

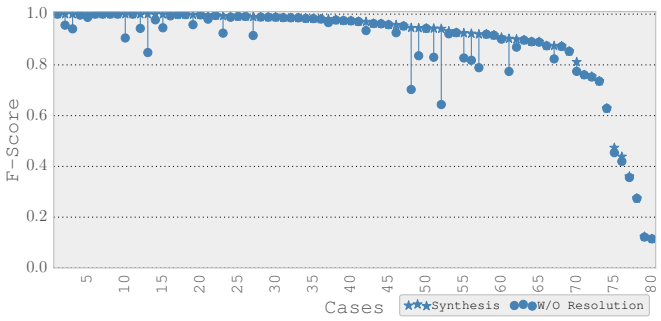


Figure 2.15: Conflict resolution improves performance.

from 0.885 to 0.878. Such improvements shows that this post-processing step is useful in removing inconsistent value pairs without affecting coverage.

Cases such as (`state` → `capital`) see the biggest improvement. These relationships tend be confused with other relationships that disagree only on a small number of values. For example, the relationship (`state` → `capital`) tends to be confused with (`state` → `largest-city`) with only minor disagreements such as `Washington` and `Olympia` vs. `Washington` and `Seattle`. These conflicting value pairs will get mixed into results because for some subset of values there may not be sufficient incompatibility to prevent merges from happening. The conflict resolution step helps to prune away such incorrect values.

Figure 2.15 compares the f-scores with and without conflict resolution. Conflict resolution improves the f-score in many cases.

We also compare our conflict resolution with majority voting. The proposed approach has a slightly higher f-score than majority voting.

2.6 Related Work

Ling and Halevy et al. studied the problem of stitching together web tables from the same domain based on column names [107]. When adapting this technique to generate mapping relationships for the whole Web, however, it tends to lead over-grouping and

low-quality mappings (as we show in the experiments), because column names are often undescriptive and too generic to be indicative of the true meanings [37] (e.g., column names like `code` and `name` are common).

Knowledge bases such as Freebase [17], and YAGO [146] curate important entity-relationships, some of which may be mapping relationships. However, the coverage of knowledge bases is low as they often miss important mappings. For instance, YAGO has none of the example mappings listed in Table 2.1, while Freebase misses two (stock and airport). For mappings that do exist in knowledge bases, there are typically no or very few synonyms such as ones listed in Table 2.6. Lastly, knowledge bases are expensive to build, yet their mappings only cover the public Web domain, and does not generalize to other domains such as enterprises.

There is a long and fruitful line of research on schema matching [132] that can suggest semantic correspondence between columns for human users. These matching relationships provide useful information about positive compatibility between tables. However, using only positive signals of compatibility are insufficient for an unsupervised algorithm to synthesize diverse tables on the web, since distinct relationships can share substantial value overlap. We introduce *negative incompatibility* specific to functional dependency observed by mapping relationships, which is shown in experiments to be critical for high-quality synthesis.

A notable branch in schema matching [20, 68, 70, 145, 185] deals with schemas extracted from Web forms collectively for matches. These techniques mainly use linguistic similarity of attribute names and distributions. However, the input schemas are required to be homogeneous and from the same conceptual domain (e.g., all forms are required to be about books, or automobiles, but not mixed). Methods in this class are the closest to our problem in the schema matching literature – we experimentally compare with a representative method from this class [70].

Compared to the traditional schema matching, there are two key aspects that differentiate our work from existing schema matching. (1) Traditional schema matching is studied in the context of a small number of database schemas. In our problem, while we also "match" semantically compatible table columns, we have to deal with millions of schemas (223M for the Web data set), which is many orders of magnitude larger than previously studied. (2) As a consequence of the scale, we can no longer afford to ask humans to verify results produced by the traditional pair-wise "match" operator ([14, 132]), which is designed to be recall-oriented with false-positives that human users are supposed manually filter out. Because in our problem pair-wise manual verification for millions of schemas is no longer feasible, we choose to group all compatible schemas and have them verified only at the group level, which would be easier and more efficient for human curation.

Although our problem would appear more difficult than schema matching, it is still tractable because we are interested in a very specific type of schemas, that are two-column tables satisfying functional-dependencies. This induces strong constraints for schema compatibility (the negative signal we exploit), which has not been explored in the classical schema matching for general tables (the existing literature mostly uses single-column type information to infer incompatibility). Furthermore, by looking at schemas holistically instead of one-pair-at-a-time, it allows us to reason globally and actually produce better matches (e.g., if table B is mostly contained by table A, and table C is also contained by A, then even if B and C share little overlap, we may still be able to group B and C using these information holistically, which may not be possible for pair-at-a-time matching).

Techniques such as the novel DataXFormer [1] represent an alternative class of approaches that "searches" tables based on user input and asks users to select relevant results to fill/join. While this is already a great improvement, our experience suggests that in many cases the need to search, retrieve, read, and manually piece together results from multiple tables is too cumbersome for this to be a viable feature in Google Doc or Microsoft Excel, where most users may not have the necessary experience to go through the full process.

Like knowledge-bases used by search engines today, we hope curating knowledge of mappings can make them easily accessible to a large number of spreadsheet users.

A related problem is to discover and validate logic rules given knowledge bases [28, 55]. Our problem is not about efficiently discovering rules that are satisfied by a monolithic knowledge base, instead we start from a large set of isolated tables and we synthesize relationships that are functional. To some extent, the relationships discovered by our technique can be used to by humans to complement and enhance existing knowledge bases.

Gupta et al. [63] build Biperpedia, which is an ontology of attributes extracted from query stream and Web text. They focus on attribute name extraction for different entity classes, but not instance values in these relationships.

Mapping tables and FDs are powerful constructs that have been studied in other contexts. For example, authors in [106, 138] study the problem of automatically inferring functional relationships using results extracted by Information-Extraction systems from a text corpus. The difficulty there is that instances extracted for the same relation may be inconsistent. For example, from sentences like "Barack Obama was born in Hawaii" and "Barack Obama was born in USA" IE systems would extract "Barack Obama" on the left, "Hawaii" and "USA" on the right, thus leading to the incorrect conclusion that the relationship of birth-place is not functional. If the results extracted by a text-pattern can be thought of as a table, then the task here is to infer if FD exists for that table, and the challenge is that values in the table may not be consistent. In comparison, we use tables where values are in most cases consistent in the same column. Our task is to go across the boundary of single tables and produce larger relations.

While separate solutions have been proposed for certain applications of mapping tables such as auto-join [72] and auto-fill [1, 176], we argue that there are substantial benefits for using synthesized mapping tables. First, mapping tables are general data assets that can benefit applications beyond auto-join and auto-fill. Synthesizing mapping tables in essence provides a unified approach to these related problems, instead of requiring a different solu-

tion for each problem. Second, without mapping tables, techniques like [72] perform heavy duty reasoning at runtime where the complexity grows quickly for large problem instances, and thus have trouble scaling for latency-sensitive scenarios. In comparison, mapping table synthesis happens offline. Applying mapping tables to online applications often reduces to table-lookup that is easy to implement and efficient to scale. Lastly, in trying to productizing auto-join and auto-fill using techniques like [72, 176], we notice that while the quality are good in many cases, they can also be unsatisfactory in others, which prevents wider adoption in commercial systems. Synthesized mapping tables, on the other hand, provide intermediate results that are inspectable, understandable, and verifiable, which are amenable to human curation and continuous user feedback. Thus it is an important problem worth studying.

CHAPTER 3

RC-INDEX: INDEX FOR RANGE QUERY RESULT DIVERSIFICATION

3.1 Introduction

Query result diversification is an important aspect of user-facing applications, such as data exploration, Web search, and recommendation systems [158, 46, 43, 186]. The need for diversification arises when the system has to limit the number of query results: since human users can visually process limited information, interfaces typically need to limit the data they display on the screen to a few points on a map or a small number of items on a list. Diversification is one common way to present representative results to users, and it is employed by many real-world systems. For example, even though there are thousands of ATMs in Manhattan, a search in Google Maps typically reveals no more than 20 at any zoom level, and the chosen locations are typically dispersed in the viewing area. Product searches in online marketplaces, such as Amazon, also employ diversity: a search for laptops in a particular price range typically yields diverse brands and models on the first page of results, rather than similar laptops from different retailers.

Our goal is to provide an efficient and scalable solution to the problem of selecting a diverse subset of the result of *general* range queries over a single relation. Systems that employ diversification typically try to optimize a specified diversity score function. The diversity score is defined over a bivariate distance function, which is domain- and application-specific, and measures the distance between any two items of a dataset. When a user issues a query, the system needs to retrieve a set S of k items such that: (1) every item in S is in the result of the query; (2) the diversity score of S is maximized. This is a

challenging problem to solve efficiently and in a scalable way, as it is practically infeasible to compute the diversity score of every k-sized subset of the query result.

Existing approaches fail to address this problem effectively. Many existing diversification techniques follow a "process-first-diversify-next" approach [89]: First, they execute the query normally to retrieve all results, and they subsequently employ appropriate algorithms to identify a subset with high diversity score. The "process-first-diversify-next" techniques have a big problem: efficiency. In practical diversification scenarios, systems typically only need to retrieve a relatively small subset of items, thus, computing the entire query result is often computationally wasteful.

Some prior work on result diversification has achieved efficient solutions at the expense of generality. For example, some work focuses on extracting diverse items from a data stream through continuous querying [118, 47, 44]. These techniques achieve efficiency by reusing prior diversification results as new items arrive. Similarly, in some data exploration scenarios, such as geolocation visualization, subsequent queries are correlated, which again allows reuse of overlapping items in query results [90, 91, 92]. These techniques do not perform well in more general cases where query results do not overlap significantly, even if the query predicates are over a fixed set of attributes. For example, if a user issues subsequent queries for "laptops under $1000", "laptops between $1000 and $1500", and "laptops above $1500", even though the queries are very similar, their results do not overlap, and thus these techniques do not apply.

In this chapter, we propose a general, index-based approach for diversifying the results of multi-dimensional range queries over a single relation. At a high level, our approach transforms each range query into a set of subordinate searches, performs subordinate searches using a novel index structure, called RC-Index, and finally extracts a diverse subset from the merged results of subordinate searches. The RC-Index achieves efficiency by retrieving a very small set of candidate items compared to "process-first-diversify-next"

techniques. Furthermore, we prove theoretically and demonstrate empirically that our method has a good approximation ratio compared to the state-of-the-art algorithms.

Our work addresses four important challenges.

• **Efficiency.** Existing techniques for addressing diversification in ad-hoc queries require the retrieval of the entire query result before applying diversification algorithms to retrieve a diverse k-sized subset. In contrast, RC-Indexes retrieve a much smaller set of candidate items (up to 99.4% reduction compared to state of the art), which leads to order of magnitude faster response times.

• **Generality.** One of our main goals in this chapter is to support general range queries over a relation. Prior work achieved efficiency only by restricting generality, through the assumption that subsequent queries share a large portion of results. In contrast, RC-Indexes do not make such assumptions. The RC-Index is a new index-based access method to relations that allows for fast retrieval of diverse subsets of results for range queries.

• **Effectiveness.** RC-Indexes provide theoretical guarantees with respect to diversification quality. The approximation ratio is based on tunable parameters, and at the limit approaches $\frac{1}{2}$, which is the optimal polynomial time ratio for our diversification problem [137].

• **Flexibility.** RC-Index use existing indexing structures as submodules to support range search. Our implementation is based on range trees, but a system designer may opt for different index structures (e.g., k-d trees) if they are better-suited for a given application.

We organize our contributions as follows.

• We define the problem of query result diversification, provide background on existing diversity score functions, discuss important properties of distance metrics, and give an overview of our solution. (Section 3.2)

• We introduce the core of our approach, a novel index structure called RC-Index, which combines range selection indexes (range trees) with diversity indexes (cover trees). We

describe the query module of our system, which uses the RC-Index to answer range queries with diversification requirements, and analyze its complexity. (Section 3.3)

• We perform a theoretical analysis to show that RC-Index is effective at providing diverse results. Specifically, the diversity score of sets returned by our system approximates the optimal diversity score by a factor of $\frac{b-1-2b^{1-\delta}}{2(b-1)}$, where $b > 1$ and $\delta \geq 1$ are parameters that control the ratio and time/space complexity. When b or δ approaches infinity, the limit of the ratio is $\frac{1}{2}$. This is the optimal polynomial approximation for our diversification problem under pseudometric distance functions [137]. (Section 3.4)

• We discuss index selection with respect to RC-Indexes, and considerations in the choice of attributes the index is built on. (Section 3.5)

• We evaluate our prototype system over real-world datasets and demonstrate that RC-Indexes are both efficient and effective at range result diversification. Specifically, we demonstrate that our approach achieves better diversity scores than the state of the art and alternative baselines, and it is substantially faster as well. Overall, RC-Indexes provide an extremely effective way to support range query diversification: our system can 10^6 items in 330 seconds and answer a query in under a second. Our approach is also more general, as it subsumes prior work, handling both streaming and relational queries. (Section 3.7)

• Finally, we discuss related work and extensions. (Sections 3.8 and 3.9)

3.2 Overview and Background

We begin this section with the definition of range query result diversification, an optimization problem over a diversity score function $f(S, dis)$. We then discuss two popular diversity score functions from prior work, and desirable properties for the domain-specific distance measures used by the diversity score functions. We continue to present an overview of our system solution in Section 3.2.3. The core of our approach is a special index structure, the RC-Index. The RC-Index is a novel combination of two types of in-

dex structures: cover trees and range trees. We describe these structures in Sections 3.2.4 and 3.2.5 before proceeding with the details of our framework in Section 3.3.

3.2.1 Result Diversification

In this section, we define the problem of selecting a diverse subset of the results of a range query. We use X to denote a database with a single relation $R(\mathbb{A})$ over the set of attributes $\mathbb{A} = \{A_1, A_2, ...\}$. A range query q can apply interval filters to a subset of attributes $\mathbb{A}_q \subseteq \mathbb{A}$. Each attribute in $\mathbb{A}_q$ has an ordered domain, and a bivariate function $dis(x_i, x_j)$ measures the distance between items $x_i, x_j \in X$. The distance function is defined over a subset of attributes $\mathbb{A}_{dis} \subseteq \mathbb{A}$, which may or may not overlap with $\mathbb{A}_q$. Given a database X and a query q, we wish to find the subset of k items in the result $q(X)$ that maximizes a diversity score function defined over dis.

Problem 3.2.1 (Range Query Result Diversification)**.** *Given a set of items $X = \{x_1, x_2, ...\}$, a bivariate distance function $dis(\cdot,\cdot)$ on X, a range query q with result set $q(X)$, and a positive integer $k \leq |q(X)|$, range query result diversification selects the subset $S \subseteq q(X)$ that maximizes a diversity score function f over dis:*

$$\begin{aligned} \max_{S} \quad & f(S, dis) \\ s.\,t. \quad & S \subseteq q(X) \\ & |S| = k \end{aligned}$$

The distance measure dis is application-specific. The diversity score function $f(S, dis)$ has two popular forms [45, 158, 43]:

$$f_{min}(S, dis) = \min_{x_i, x_j \in S \,\wedge\, x_i \neq x_j} \{dis(x_i, x_j)\}$$

$$f_{sum}(S, dis) = \sum_{x_i, x_j \in S \,\wedge\, x_i \neq x_j} dis(x_i, x_j)$$

The first form, f_{min}, computes the minimum distance of the items in S; the corresponding diversification problem is called MAXMIN. The second form, f_{sum}, computes the sum of pairwise distances of items in S; the corresponding diversification problem is called MAXSUM. Both MAXMIN and MAXSUM are NP-hard [51, 147, 43].

In practice, the diversification score is used to select appropriate top-k items. We demonstrate its application with an example:

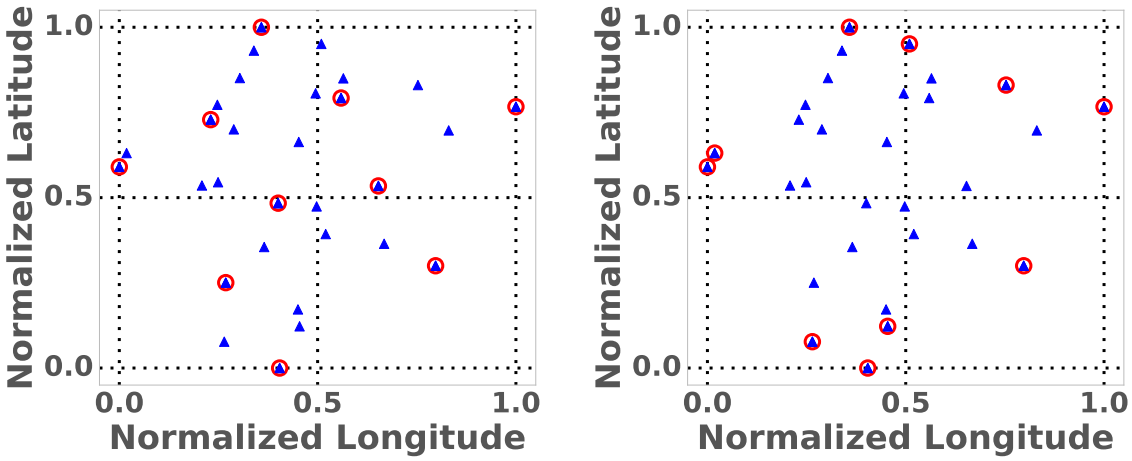
Example 3.2.2. *A user queries a dataset of ATM information to find those with closing time after 8pm. The application displays 10 of the ATMs in the result with diverse locations based on their Euclidean distance $dis_E(\cdot,\cdot)$, using the following query:*

```
SELECT *
FROM ATM_data
WHERE Close_time >= 20
LIMIT 10 DIVERSE(disE(Latitude, Longitude)) MAXMIN;
```

This query, q, applies a range filter to one attribute ($\mathbb{A}_q = \{$`Close_time`$\}$). The `LIMIT` *clause specifies that the query will return 10 items. Normally, the* `LIMIT` *clause would return any 10 items, but in this case, the clause is augmented with a diversification objective: We want the set of 10 results that maximizes the* MAXMIN *diversity score defined over the Euclidean distance $dis_E(\cdot,\cdot)$ on attributes* `Latitude` *and* `Longitude` *($\mathbb{A}_{dis} = \{$`Latitude`, `Longitude`$\}$).*

Figure 3.1a illustrates the result of the query of Example 3.2.2 over a small sample of the ATM dataset. All ATMs locations that satisfy the range predicate are denoted with blue triangles; the red circles denote the selected 10 diverse locations. Figure 3.1b shows the result of the same query with the MAXSUM diversity score.

We can see that solution under f_{min} provides better coverage [46, 47]. So we focus on f_{min}, i.e., MAXMIN, in the rest of this chapter. We extend it to MAXSUM in Appendix 3.6.



(a) $f_{min}(S, dis_E) = \min\{dis_E(x_i, x_j)\}$ (b) $f_{sum}(S, dis_E) = \sum dis_E(x_i, x_j)$

Figure 3.1: Diversification on a sample of ATMs in New York City under two diversity score functions. We select $k = 10$ ATMs (red circles), out of a total of 30 (blue triangles) in each figure. The solution under f_{min} has better coverage than the one under f_{sum}.

3.2.2 Distance Function

The distance function is an important component of diversification, as the problem objective involves the maximization of pairwise distances of items in S. The distance function is domain- and application-specific, and can involve any of the attributes.

Intuitively, one can cluster X according to the distance metric $dis(\cdot,\cdot)$, and use this clustering to solve Problem 3.2.1. However, the challenge is that the attributes over which the distance is defined may or may not overlap with the attributes filtered by q: In Example 3.2.2, the distance is defined over the *latitude* and *longitude* attributes ($\mathbb{A}_{dis} = \{\texttt{Latitude}, \texttt{Longitude}\}$), while the range condition is over the `Close_time` attribute ($\mathbb{A}_q = \{\texttt{Close\_time}\}$).

While there are no general restrictions on the distance metric with respect to the definition of Problem 3.2.1, some properties of the distance metric affect the problem complexity. Problem 3.2.1 under MAXMIN is APX-hard for general distance functions [137]. This means that no polynomial time algorithm can provide a performance guarantee better

than a constant ratio unless P=NP. However, when the distance function is symmetric and satisfies the triangle inequality, a simple greedy heuristic provides a $1/2$-approximation; no polynomial algorithm can provide a performance guarantee better than $1/2$ unless P=NP [147, 137]. Following prior work in this area [147, 137, 47, 44], we also assume the distance function should satisfy symmetry and triangle inequality in this chapter.

Definition 3.2.3 (Distance Function)**.** *The bivariate distance function* $dis : X \times X \to \mathbb{R}_{\geq 0}$ *in Problem 3.2.1 must be a pseudometric satisfying the following properties:*

$dis(x,x) = 0$

$dis(x,y) = dis(y,x)$ *(Symmetry)*

$dis(x,y) + dis(y,z) \geq dis(x,z)$ *(Triangle Inequality)*

Many common distance functions used in practice satisfy Definition 3.2.3. Here we list a few:

- The metrics induced by any L_p-norm with $p \geq 1$. These include Manhattan distance (L_1-norm) and Euclidean distance (L_2-norm).
- Graph metric. This is based on a graph with vertices and edges. The distance between two vertices is the number of edges in the shortest path connecting them.
- The "diversity ordering"-based distance [157]. This distance function defines a total ordering of the attributes like *car make* $\prec$ *car model* $\prec$ *color* $\prec$ *year*, which means *car make* has higher priority than *car model* does and so on. The distance between two items is greater if the two items differ on a higher priority attribute.

3.2.3 Solution and System Overview

Our framework has two modules as illustrated in Figure 3.2: Query Module and Diversification Module. The Query Module builds an index, RC-Index, offline to support range queries. Then when a query arrives, the Query Module uses the index to extract a set of candidates and passes this set to the Diversification Module. Finally, this Diversification

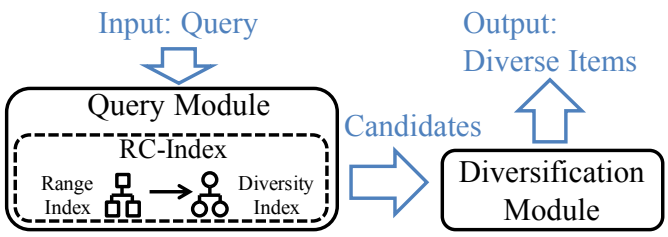


Figure 3.2: Solution overview. Our approach involves two modules: Query Module and Diversification Module.

Algorithm 5: Query Evaluation

Input: Range query q; Parameter k; RC-Index RC; Extra level δ.
Output: Set of diverse items S.

1 $T \Leftarrow$ `RangeQueryTreeExtraction`(q,k,RC,δ) `// QM: Algo. 7`
2 $X^C \Leftarrow$ `CandidateExtraction`(k,T,δ) `// QM: Algo. 6`
3 $S \Leftarrow$ `GreedyDiversification`(X^C,k) `// DM: Algo. 8`

Module finds k representative items to present to the user. Algorithm 5 shows the pipeline of query evaluation at a high level. Line 1 and Line 2 belong to Query Module (QM). Line 3 is Diversification Module (DM). We will elaborate in Section 3.3.

The key to the success of this approach is that we limit the number of candidates extracted by the Query Module but ensure their diversity. On the one hand, we significantly reduce the number of candidates to save much evaluation time. On the other hand, we design a special index in the Query Module to ensure high diversity of these few candidates, which eventually guarantee a good final result.

The special index, RC-Index, in the Query Module combines two types of indexes: Range Index and Diversity Index. The Range Index is to support range queries. It can be a B+ tree (for 1-dimensional queries), interval tree, R-tree, VA-file, k-d tree, range tree, and so on. Each Diversity Index is a cover tree built on a subset of items. It organizes the items according to their pairwise distances. Items near root are far from each other while items near bottom are close to each other. It can help us limit the number of candidate items we

extract, while ensuring the diversity of candidates. We will prove this desired property in Section 3.4.1.

3.2.4 Cover Tree

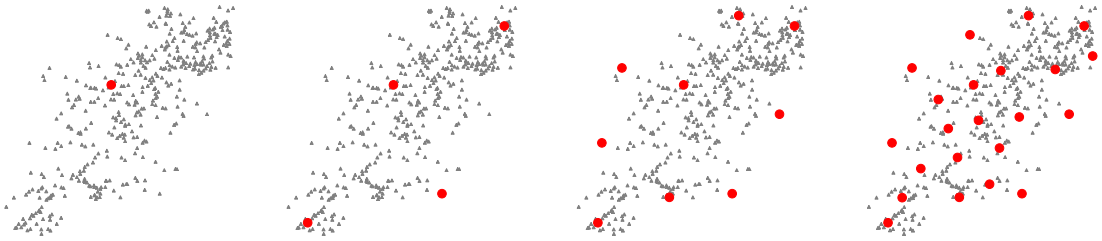
A cover tree [15] is a data structure that naturally diversifies items. It is similar to navigating nets [97], which were originally designed for nearest neighbor search. Every level of a cover tree has a distance threshold. The distance between every pair of items at this level must be greater than this threshold. The threshold decreases from root to leaf. So, items at a higher level (i.e., closer to the root) are farther away from each other. Intuitively, one can use a cover tree over a dataset X to retrieve k diverse items from X by selecting any k items from the highest level that contains k or more items. Drosou and Pitoura [47] used cover trees for diversification. However, that work assumes queries over continuous data with sliding windows, so a single cover tree can answer consecutive queries that share results. In this chapter, we are looking at a more general problem, as we wish to support range queries that may or may not share results. Our RC-Index uses cover trees internally, but it is a more complex structure, and our algorithms and approximation guarantees differ from the ones in the prior work.

Formally, a cover tree embeds items into a tree with multiple levels. Each level has an integer level number, ℓ. The root is at the highest level, $\ell_{\max}$. Each level also has a distance threshold $\theta_\ell = b^\ell$, where $b > 1$ is a "base" distance parameter defined for each cover tree. Let C_ℓ be the set of items at Level ℓ. A cover tree must obey the following three invariants:

1. **Nesting**: $C_\ell \subseteq C_{\ell-1}$. If an item appears at level ℓ, it must appear at all levels below ℓ.

2. **Covering**: If $x_i \in C_\ell$ and x_j is its direct child, $dis(x_i, x_j) \leq \theta_\ell = b^\ell$. This implies that an item at Level ℓ covers all its direct children within a ball whose radius is θ_ℓ.

3. **Separation**: If $x_i, x_j \in C_\ell$ and $x_i \neq x_j$, $dis(x_i, x_j) > \theta_\ell = b^\ell$. This indicates that the pairwise distances between all distinct items at Level ℓ must be greater than θ_ℓ. In



(a) $\ell_{\max} = 0$ (b) $\ell = -1$ (c) $\ell = -2$ (d) $\ell = -3$

Figure 3.3: The highest four levels of an example cover tree on ATMs in New York City with normalized coordinates. Items at each level ℓ are highlighted in red. The tree satisfies Nesting, Covering, and Separation.

practice, items with $dis(x_i, x_j) = 0$ should be aggregated as a list of ids at all levels to satisfy the separation invariant.

We should note that, conceptually, the level number ℓ of a cover tree goes from $\ell_{\max}$ to $-\infty$. When ℓ is very small, C_ℓ is all items, each of which has itself as the only child at $C_{\ell-1}$. However, we still need only $O(n)$ space to store all items of a cover tree where n is the number of *distinct* items. We will defer the discussion of cover tree's time and space complexity to Section 3.4.

Figure 3.3 depicts the highest four levels of an example cover tree on a set of ATMs in New York City (south of Central Park). The ATMs' latitude and longitude are normalized to $[0, 1)$. We highlight items at each level as red points and display all other items as grey points. We set the base distance of this cover tree as $b = 2.0$, so the root of the cover tree turns out to be at Level $\ell_{\max} = 0$. **Nesting:** every lower level contains all items at higher levels. For example, the root appears in all levels in the figure. **Covering:** the distance between a parent item and a child item must be within the distance threshold of the parent's level. For example, the distances between the root and all its direct children are no more than $\theta_{\ell_{\max}} = b^{\ell_{\max}} = 1.0$. **Separation:** the pairwise distances between all items at each level

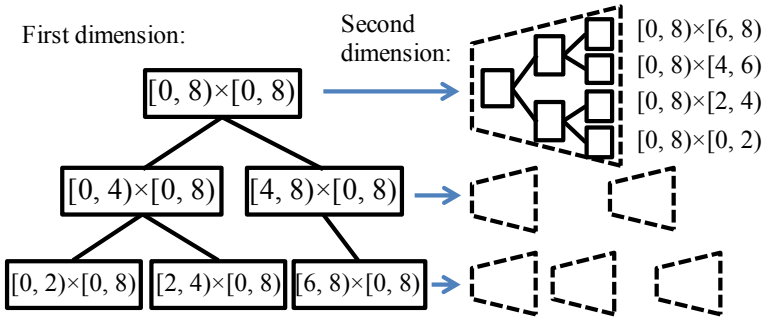


Figure 3.4: An example 2-dimensional range tree. Every node in the first dimension range tree points to another range tree in the second dimension.

are greater than the corresponding threshold. For example, at Level $\ell = -1$ where $\theta_\ell = 0.5$, the pairwise distance between items are greater than 0.5.

3.2.5 Range Tree

A range tree [111, 12, 166, 100] is a nested tree structure. The left hand side of Figure 3.5 shows an example of a simple, 1-dimensional range tree. The root of the tree represents the entire range and every descendant non-leaf node corresponds to a subrange. The leaf nodes are the actual data items. Figure 3.4 illustrates a 2-dimensional range tree. In this particular example, we assume the data space is $[0,8) \times [0,8)$ and each inner range is evenly split into two subranges to simplify the presentation. In practice, the actual space and separator depends on the data distribution. We also omit leaf items for simplicity. The root of the whole range tree is $[0,8) \times [0,8)$ at the left side. It splits on the first dimension to get the two children $[0,4) \times [0,8)$ and $[4,8) \times [0,8)$. In the meantime, it points to another range tree at the right hand side. This range tree splits on the second dimension till the finest subrange like $[0,8) \times [0,2)$ and $[0,8) \times [2,4)$. Similarly, every inner node in the first dimension points to a range tree that splits on the second dimension. So when we implement a range tree, each node can have at most three children in two categories: $left$ and *right* for the current dimension and *next* for the next dimension. A node at the finest subrange does not have $left$ and *right*. A node in the last dimension does not have *next*.

3.3 Index-based Framework

In this section we focus on how the Query Module and the Diversification Module work. We prove their complexity and approximation ratio in Section 3.4.

3.3.1 Query Module: Sketch

We start by explaining how we devise the Query Module from scratch, hopefully providing some insight of the entire work here before diving into the details.

The fundamental question in Problem 3.2.1 is how to extract diverse items from a large set of input items. We first notice that a Diversity Index (DI), which organizes items according to their distances, is required. A cover tree meets the requirement, but one single cover tree is not enough to answer various user queries. For example, if we build only one cover tree $CT_{[0,8)}$ for items in range $[0,8)$, how can we answer a query on $[2,8)$? Remember, as we point out in Section 3.2.2, the attributes used to compute distance (e.g. *latitude* and *longitude*) can be different from the attributes filtered by a range query (e.g. *close time*). So we cannot assume a cover tree's structure follows any range pattern. Specifically, we cannot assume one or more subtrees of $CT_{[0,8)}$ cover and only cover items in range $[2,8)$.

So we build multiple cover trees on different partitions of X and extract diverse items from them *with performance guarantee*, which is one of our main contributions. We base our approach on two important techniques. First, we can carefully extract items from multiple non-overlapping cover trees to ensure the diversity score be no less than a factor times the optimal diversity score. In other words, this approach is a constant-factor approximation algorithm (Section 3.3.2). Second, we answer various queries while limiting the number of cover trees we build. This is because we transform all range queries to a limited number of subordinate searches. For example, we may change a 1-dimensional range query on $[2,8)$ into two subordinate searches on $[2,4)$ and $[4,8)$. So we only need to build cover trees for these subordinate searches. These two techniques together form an index to sustain our Query Module. We call it RC-Index.

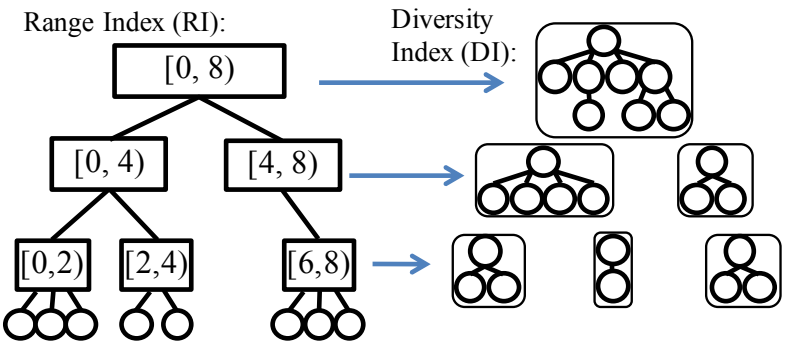


Figure 3.5: RC-Index: maps every Range Index (RI) node to a Diversity Index (DI) tree. It is the core of Query Module. This example Range Index supports 1-dimensional range queries.

Example 3.3.1 (Continuing Example 3.2.2)**.** *A user can create RC-Index as follows to support query q in Example 3.2.2:*

```
CREATE RC_Index ON X(Close_time)
DISTANCE disE(Latitude, Longitude);
```

Figure 3.5 depicts the high level idea of RC-Index of our Query Module. It consists of two indexes: Range Index (RI) and Diversity Index (DI). The RI is to support the range query. It can be a B+ tree, R-tree, k-d tree, range tree, and so on. On the left hand side in the figure, we exhibit an example RI for 1-dimensional range queries. We display a range on each node to indicate its coverage. The root covers the range $[0,8)$. Then it splits the range into sub-ranges as children until the range is small enough. All these ranges correspond to inner nodes, while the leaves are actual items.

Every RI node is mapped to a DI tree on the right hand side. The root of the RI is mapped to a DI tree that covers all the eight items. Then the RI node $[0,4)$ is mapped to a DI tree that covers all five items within this range. Similarly, all inner nodes of RI are mapped to a corresponding DI tree. At the bottom, every individual item is itself a DI tree, so we do not need to explicitly map leaves of RI to DI, which saves some space.

Our Query Module has the following features.

• As we mentioned before, we allow the attributes used for distance calculation $\mathbb{A}_{dis}$ to be different from the attributes filtered by the range query $\mathbb{A}_q$, which makes our approach applicable to more applications.

• Our approach supports not only any range query q on the query attributes $\mathbb{A}_q$ but also various k. For instance, if a system such as Amazon would like to return top 10 results to one user but top 20 results to another user due to customized settings, our approach can do it without rebuilding the indexes.

• Our approach works in a dynamic scenario where a system inserts and deletes items between queries.

3.3.2 Query Module: Diversity Index

We introduce Diversity Index first as it is the essential part of our approach. We use the cover tree [15] which is originally designed for nearest neighbor search. It organizes items according to distance. So we can extract diverse items from one or more cover trees with a performance guarantee.

Intuitively, we can extract diverse items top-down from a single cover tree. But one cover tree is not capable of answering various range queries. So we build multiple cover trees which together cover the items to be diversified, i.e. $q(X)$, and do not contain any extra items (Section 3.3.3). We extract more than k candidate items from these cover trees in our Query Module. Later we choose exactly k diverse items from the candidates in our Diversification Module (Section 3.3.5).

Algorithm 6 explains how we extract more than k candidate items from multiple cover trees. We enumerate the cover trees and extract a level of items from each tree in Function `ExtractTree` (from Line 5 to Line 10). This function finds the highest level with at least k items (Line 8) and goes δ levels down (Line 9) to return all items at that level. Level $(\ell_k - \delta)$ always exists because the level number of a cover tree goes from $\ell_{\max}$ to $-\infty$.

Algorithm 6: Candidate Extraction

Input: Parameter k; Set of cover trees $T = \{CT_1, CT_2, ...\}$; Extra level δ.
Output: Set of candidate items X^C.

$X^C \Leftarrow \emptyset$
foreach $CT \in T$ **do**
  $X^C \Leftarrow X^C \cup \texttt{ExtractTree}(CT, k, \delta)$
return X^C
Function `ExtractTree`(CT, k, δ)
  if $|CT| \leq k$ **then**
    return All items in CT
  $\ell_k \Leftarrow \arg\max_{\ell} |C_\ell| \geq k$
  $\ell \Leftarrow \ell_k - \delta$
  return C_ℓ

Example 3.3.2. *Consider applying Algorithm 6 to one single cover tree in Figure 3.3 with parameters $k = 3$ and $\delta = 1$. Our function `ExtractTree` firstly makes sure this tree has more than 3 items. Then it finds the highest level with at least 3 items: Level $\ell_k = -1$. Finally it goes $\delta = 1$ level down to Level -2 and returns all its 9 items.*

The candidate items extracted by Algorithm 6 are diverse, which allows us to later select k items with high diversity score from these candidates. Formally, if the optimal diversity score on X is f^*, and the optimal diversity score on extracted X^C is f^C, we have $f^C \geq \frac{b-1-2b^{1-\delta}}{b-1} \cdot f^*$. We prove this property in Section 3.4.1.

3.3.3 Query Module: Range Index

We use *Range Tree* [111, 12, 166, 100] to transform *any* range query to $\log^d n$ subordinate searches, where n is the size of the data and $d = \mathbb{A}_q$ is the dimensionality. Range tree is one of the data structures that efficiently support range query [13]. We utilize range tree as our Range Index for two reasons: (1) It can help us transform a range query to a reasonable number of subordinate searches. Given any range query, we visit at most $\log^d n$ nodes in a range tree, each of which can be viewed as a subordinate search. Then we can perform each subordinate search with the help of a Diversity Index. (2) It can answer a range query efficiently. In a conventional range query evaluation scenario, one can extract

Algorithm 7: Range Query Tree Extraction

Input: Range query q; Parameter k; RC-Index RC; Extra level δ.
Output: Set of cover trees T.

```
1  T ⇐ QueryRangeTree(q, RC.root)
2  return T
3  Function QueryRangeTree(q, node)
4      if node.range ⊆ q.range then
5          return {node.CT}
6      if node.range ∩ q.range = ∅ then
7          return ∅
8      nowT ⇐ ∅
9      foreach child ∈ {node.left, node.right, node.next} do
10         nowT ⇐ nowT ∪ QueryRangeTree(q, child)
11     return nowT
```

the entire $q(X)$ with time complexity $O(\log^d n + |q(X)|)$. So it is faster than k-d tree [11] or quad tree [53] whose worst case query complexity is $O(d \cdot n^{1-1/d} + |q(X)|)$ [99].

Given a range query, we can answer the query by visiting at most $\log^d n$ nodes. Let's see an example on a 2-dimensional query.

Example 3.3.3. *When receiving a 2-dimensional query* $[2,8) \times [2,6)$ *on a range tree in Figure 3.4, we split the first dimension and stop at* $[2,4) \times [0,8)$ *and* $[4,8) \times [0,8)$. *Then we split the second dimension to reach four nodes* $[2,4) \times [2,4)$, $[2,4) \times [4,6)$, $[4,8) \times [2,4)$, *and* $[4,8) \times [4,6)$.

We briefly present the time and space complexity of a conventional range tree. The query complexity is $O(\log^d n)$, because the range of each dimension is split to $\log n$ sub-ranges. Its batch construction time is $O(n \log^d n)$. The amortized time complexity of insertion and deletion is $O(\log^d n)$ when we carefully maintaining the balance of the tree [166, 100, 112]. Its space complexity is $O(n \log^d n)$.

3.3.4 Query Module: Candidate Extraction

We map every range tree node to a cover tree to form our RC-Index. So instead of retrieving all data items of range tree nodes, we query the corresponding cover trees to

extract a few diverse items as candidates. Algorithm 7 depicts how we traverse the RC-Index to collect a set of cover trees for a range query. We later pass the cover trees to Algorithm 6 to get candidates. Here is an example:

Example 3.3.4. *(Continuing Example 3.3.3) Given the range query* $[2,8) \times [2,6)$*, we traverse our index and stop at four nodes:* $[2,4) \times [2,4)$*,* $[2,4) \times [4,6)$*,* $[4,8) \times [2,4)$*, and* $[4,8) \times [4,6)$*. We collect their corresponding cover trees into a set* T*. Then we pass it to Algorithm 6 to get candidate items* X^C*.*

This approach successfully reduces the number of scanned items from $|q(X)|$, i.e. $O(n)$ in the worst case, to $O(k\gamma^{4(\delta+1)}\log^d n)$ in the worst case as we prove in Section 3.4.1.

3.3.5 Diversification Module

This module takes a set of candidate items $X^C = \{x_1, x_2, ...\}$ as input and output exactly k diverse items. Since the number of candidates is limited by the previous Query Module, we deploy a simple $O(k \cdot |X^C|)$ greedy algorithm here [147, 137]. Notice that this module does not stick to any particular algorithm. One can change the algorithm here to any other algorithms when necessary.

Algorithm 8 introduces the detail of this greedy algorithm. Initially, it selects a random item from X^C to put into the output set S (Line 2). Then it maintains an array to track the distances between selected items and unselected items (Line 5). The algorithm iteratively chooses the item with the greatest distance from the selected items until it finds k items (Line 6 to 11). If we want to solve the MAXSUM version of this problem, we change Algorithm 8 to another one as we explain in Section 3.6.

The time complexity of Algorithm 8 is $O(k \cdot |X^C|)$. The space complexity is $O(|X^C|)$. It is a 1/2-approximation algorithm [147, 137].

Algorithm 8: Greedy Diversification

Input: Candidate set $X^C = \{x_1, x_2, ...\}$; Size of output k where $k \leq |X^C|$.
Output: Set of diverse items S.

$x_{random} \Leftarrow$ one random item in X^C
$S \Leftarrow \{x_{random}\}$
$X^C \Leftarrow X^C - \{x_{random}\}$
foreach $x_i \in X^C$ **do**
    $dist[x_i] \Leftarrow dis(x_i, x_{random})$
while $|S| < k$ **do**
    $x_{farthest} \Leftarrow \underset{x_i \in X^C}{arg\,max}\ dist[x_i]$
    $S \Leftarrow S \cup \{x_{farthest}\}$
    $X^C \Leftarrow X^C - \{x_{farthest}\}$
    foreach $x_i \in X^C$ **do**
        $dist[x_i] \Leftarrow \min\{dist[x_i], dis(x_i, x_{farthest})\}$
return S

Query	Approximation Ratio:	$\frac{b-1-2b^{1-\delta}}{2(b-1)}$
	Space Complexity:	$O(k\gamma^{A(\delta+1)}\log^d n)$
	Time Complexity:	$O(k^2\gamma^{A(\delta+1)}\log^d n)$
Index	Batch Construction Time:	$O(\gamma^6 n\log^{d+1} n)$
	Amortized Insert/Delete:	$O(\gamma^6 d\log^{d+2} n)$
	Space Complexity:	$O(n\log^d n)$

Figure 3.6: Performance.

3.4 Performance Analysis

We analyze the performance of query evaluation and index construction in this section. Figure 3.6 summarizes it.

3.4.1 Query Quality and Complexity

We prove the approximation ratio, time complexity, and space complexity for answering a query using Algorithm 5.

3.4.1.1 Approximation Ratio

Given a set of cover trees that covers items in query result $q(X)$, we prove that Algorithm 6 extracts a high-quality set of candidate items X^C:

Theorem 3.4.1. *Let the optimal diversity score on $q(X)$ be f^* and the optimal diversity score on extracted X^C be f^C. We have $f^C \geq \frac{b-1-2b^{1-\delta}}{b-1} \cdot f^*$, where b is the base distance parameter of the cover trees we build and δ is the extra level parameter of Algorithm 6.*

In order to prove this theorem, we show there exists a subset $Y = \{y_1, ..., y_k\} \subseteq X^C$ whose diversity score f^Y is at least $\frac{b-1-2b^{1-\delta}}{b-1} \cdot f^*$. Specifically, let the optimal set be $S^* = \{x_1, ..., x_k\} \subseteq q(X)$ which leads to the optimal diversity score f^*. These $\{x_1, ..., x_k\}$ may come from one or more cover trees. Now we define $Y = \{y_1, ..., y_k\}$ based on S^*. Each y_i can be viewed as a "substitute" of x_i on x_i's cover tree CT:

• Case 1: If the tree CT has no more than k items, Algorithm 6 puts all items of this tree into X^C. So we define $y_i = x_i \in X^C$.

• Case 2: If the tree CT has more than k items, we define y_i as the ancestor of x_i at Level $(\ell_k - \delta)$, where ℓ_k is the highest level of tree CT with at least k items (Line 8). This y_i is also returned in X^C.

We would like to show x_i and y_i are close enough so that f^Y is not much worse than f^*. Formally, the distance between x_i and y_i satisfies the following lemma:

Lemma 3.4.2. $dis(x_i, y_i) \leq \frac{b^{-\delta}}{1-b^{-1}} \cdot f^*$.

Proof. Similarly, $dis(x_i, y_i)$ has two cases:

• Case 1: When the tree CT has no more than k items, since $y_i = x_i$, $dis(x_i, y_i) = 0 \leq \frac{b^{-\delta}}{1-b^{-1}} \cdot f^*$.

• Case 2: When the tree CT has more than k items, any k items at Level ℓ_k also exist at Level $(\ell_k - \delta)$ due to **Nesting** and should be extracted. They have already formed a solution with diversity score b^{ℓ_k} because of **Separation**. In addition, as f^* is the optimal diversity score of all items, $b^{\ell_k} \leq f^*$.

Since y_i is the ancestor of x_i and y_i is at Level $(\ell_k - \delta)$, according to **Covering**,

$$dis(x_i, y_i) < \sum_{\ell=-\infty}^{\ell_k-\delta} b^{\ell} = \frac{b^{\ell_k-\delta}}{1-b^{-1}}$$

So

$$dis(x_i, y_i) < \frac{b^{\ell_k-\delta}}{1-b^{-1}} = \frac{b^{-\delta}}{1-b^{-1}} \cdot b^{\ell_k} \leq \frac{b^{-\delta}}{1-b^{-1}} \cdot f^*$$

□

Now we can prove Theorem 3.4.1:

Proof. According to triangle inequality and symmetry (Definition 3.2.3), for any x_i and the corresponding y_i:

$$dis(x_i, x_j) \leq dis(x_i, y_i) + dis(y_i, y_j) + dis(x_j, y_j)$$

Since f^* is the diversity score of $\{x_1, ..., x_k\}$, $dis(x_i, x_j) \geq f^*$. So for any distinct y_i and y_j:

$$\begin{aligned} dis(y_i, y_j) &\geq dis(x_i, x_j) - dis(x_i, y_i) - dis(x_j, y_j) \\ &\geq f^* - \frac{b^{-\delta}}{1-b^{-1}} \cdot f^* - \frac{b^{-\delta}}{1-b^{-1}} \cdot f^* \\ &= (1 - \frac{2b^{-\delta}}{1-b^{-1}}) \cdot f^* \\ &= \frac{b-1-2b^{1-\delta}}{b-1} \cdot f^* \end{aligned}$$

Since $f^C \geq f^Y \geq dis(y_i, y_j)$, $f^C \geq \frac{b-1-2b^{1-\delta}}{b-1} \cdot f^*$. □

Note that $\frac{b-1-2b^{1-\delta}}{b-1}$ must be greater than 0 to ensure a non-trivial bound. For instance, when $b = 2.0$ and $\delta = 3$, the bound is $1/2$. When $b = 3.0$ and $\delta = 2$, the bound is $2/3$.

We further prove the bound in Theorem 3.4.1 is tight.

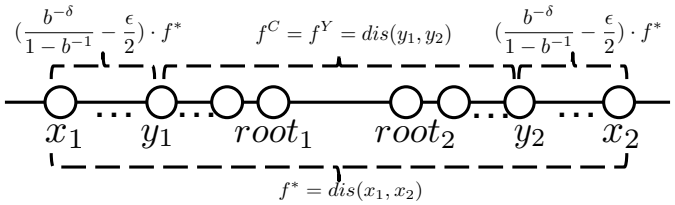


Figure 3.7: The bound in Theorem 3.4.1 is tight in the worst case.

Theorem 3.4.3. *Given any $\varepsilon > 0$, there exists a worst case making $(\frac{b-1-2b^{1-\delta}}{b-1} + \varepsilon) \cdot f^* \geq f^C$.*

Proof. We prove it by providing a worst case example (Figure 3.7). In this case, $k = 2$ and the distance function is Euclidean. All items are on a straight line. Items from x_1 to $root_1$ belong to one cover tree rooted at $root_1$. Items from $root_2$ to x_2 belong to another cover tree rooted at $root_2$.

As illustrated, there are finite number of items between x_1 and y_1 leading to $dis(x_1, y_1) = (\frac{b^{-\delta}}{1-b^{-1}} - \frac{\varepsilon}{2}) \cdot f^*$. There are also δ items between y_1 and $root_1$, each of which is at a different level. The other cover tree on the right is symmetric.

So the extracted X^C will be all items between y_1 and y_2. The optimal diversity score of X^C is therefore $f^C = dis(y_1, y_2) = (\frac{b-1-2b^{1-\delta}}{b-1} + \varepsilon) \cdot f^*$ □

Now we conclude the query result quality of Algorithm 5:

Theorem 3.4.4. *The approximation ratio of Algorithm 5 is $\frac{b-1-2b^{1-\delta}}{2(b-1)}$.*

Proof. Algorithm 5's approximation ratio is the product of the two module's approximation ratios. The Query Module's ratio is $\frac{b-1-2b^{1-\delta}}{b-1}$ as in Theorem 3.4.1. The Diversification Module's ratio is $\frac{1}{2}$. So the ratio of Algorithm 5 is $\frac{b-1-2b^{1-\delta}}{b-1} \cdot \frac{1}{2} = \frac{b-1-2b^{1-\delta}}{2(b-1)}$. □

Note that the approximation ratio approaches $1/2$ as b increases or δ increases.

This bound is also tight because the approximation bounds of both modules are tight.

3.4.1.2 Time Complexity

A cover tree's complexity analysis requires a data-dependent *expansion constant*, γ. Given X, let the closed ball of radius r centered at x be $B(x,r) = \{x' \in S | dis(x,x') \leq r\}$. So expansion constant is the smallest γ such that $|B(x, b \cdot r)| \leq \gamma |B(x,r)|$ for every $x \in X$ and $r > 0$ [15].

Each item can have at most γ^4 children [15]. Remember the number of items at Level $(\ell_k + 1)$ is strictly less than k according to Algorithm 6. So the number of items at Level $(\ell_k - \delta)$ is at most $k\gamma^{4(\delta+1)}$. Therefore we extract $O(k\gamma^{4(\delta+1)})$ items and the time complexity is also $O(k\gamma^{4(\delta+1)})$ on one single cover tree.

We have the following conclusion on multiple cover trees:

Lemma 3.4.5. *The Query Module (Algorithm 7 and 6) returns a candidate item set X^C with $O(k\gamma^{4(\delta+1)} \log^d n)$ items.*

Proof. A range query visits $O(\log^d n)$ nodes in the range tree. We map each node to a cover tree. For each cover tree, we extract $O(k\gamma^{4(\delta+1)})$ items. So we extract $O(k\gamma^{4(\delta+1)} \log^d n)$ candidate items totally. □

Lemma 3.4.6. *The worst-case time complexity of the Query Module (Algorithm 7 and 6) is $O(k\gamma^{4(\delta+1)} \log^d n)$.*

Lemma 3.4.6 follows from Lemma 3.4.5.

Theorem 3.4.7. *The worst-case time complexity of Algorithm 5 is $O(k^2\gamma^{4(\delta+1)} \log^d n)$.*

Proof. The time complexity is the sum of the complexities of the Query Module (Algorithm 7 and 6) and the Diversification Module (Algorithm 8). The complexity of the Query Module is in Lemma 3.4.6. The complexity of the Diversification Module is $O(k \cdot |X^C|)$, which equals $O(k^2\gamma^{4(\delta+1)} \log^d n)$ because $|X^C| = O(k\gamma^{4(\delta+1)} \log^d n)$ based on Lemma 3.4.5.

So the worst-case time complexity is dominated by Algorithm 8, $O(k^2\gamma^{4(\delta+1)} \log^d n)$.

□

Algorithm 9: RC-Index Batch Construction

Input: Set of items X; Dimensionality d.
Output: RC-Index *RC*.

```
1  RC.root ⇐ Construct(1, X.sortByDimension(1))
2  return RC
3  Function Construct(nowD, sortedX)
4      if |sortedX| = 0 then
5          return NULL
6      node.left ⇐ Construct(nowD, firstHalf(sortedX))
7      node.right ⇐ Construct(nowD, secondHalf(sortedX))
8      if nowD < d then
9          nextX ⇐ sortedX.sortByDimension(nowD + 1)
10         node.next ⇐ Construct(nowD + 1, nextX)
11     node.CT ⇐ construct cover tree on sortedX
12     return node
```

The exponential term $O(\gamma^{4(\delta+1)})$ looks expensive. But one should notice that cover tree is designed for nearest neighbor search, and one search query takes $O(\gamma^{12}\log n)$ time [15], whose exponent of γ is also large. In practice, this exponential term turns out to be insignificant. We demonstrate it by showing the wall-clock runtime in Section 3.7.

3.4.1.3 Space Complexity

During query evaluation, we need to store the candidates we extract. So the space complexity is $O(k\gamma^{4(\delta+1)}\log^d n)$ as in Lemma 3.4.5.

3.4.2 Index Complexity

We discuss the batch construction time complexity, insertion/deletion time complexity, and space complexity of RC-Index. We show that the RC-Index can be created and maintained efficiently, proving formal bounds on the cost of key operations

3.4.2.1 Batch Construction

Algorithm 9 shows how we batch construct RC-Index. Given a node, it builds a cover tree for the node, construct its $left$ and *right* children in the same dimension, and construct its *next* child in the next dimension.

Theorem 3.4.8. *The batch construction of RC-Index on n item takes $O(\gamma^6 n \log^{d+1} n)$ time in the worst case.*

Proof. We prove it through induction.

• When $d = 1$, we build a 1-dimensional range tree, each node of which is a cover tree. We first sort all items into an array in $O(n \log n)$ time so that we can identify the items of each inner node in linear time [1]. Then we use divide and conquer algorithm to build the RC-Index. At each node, we find the median using the sorted array, divide the items into two children, build RC-Index for each child, and finally construct a cover tree for the current node. We know that building a cover tree on n items takes $O(\gamma^6 n \log n)$ time. So we have the following recurrence relation:

$$t(n) = 2 \cdot t(n/2) + O(\gamma^6 n \log n)$$

According to the master theorem, $t(n) = O(\gamma^6 n \log^2 n)$. The overall time complexity $T_1(n)$ including sorting is still $T_1(n) = O(n \log n) + t(n) = O(\gamma^6 n \log^2 n)$.

• Suppose Theorem 3.4.8 holds when $d = m$, so $T_m(n) = O(\gamma^6 n \log^{m+1} n)$. Now we prove it also holds when $d = m + 1$.

We construct a $(m+1)$-dimensional RC-Index by recursively splitting the first dimension and building an m-dimensional RC-Index for each node. Similarly, we also sort the items by the first dimension with $O(n \log n)$ time. Then we divide and conquer to build each child. Finally, we construct a cover tree for the current node. So we have the recurrence relation:

$$\begin{aligned} t(n) &= 2 \cdot t(n/2) + T_m(n) + O(\gamma^6 n \log n) \\ &= 2 \cdot t(n/2) + O(\gamma^6 n \log^{m+1} n) + O(\gamma^6 n \log n) \\ &= 2 \cdot t(n/2) + O(\gamma^6 n \log^{m+1} n) \end{aligned}$$

[1] This is faster than recursively applying the $O(n)$ selection algorithm in practice.

Algorithm 10: RC-Index Insertion

Input: New item x; RC-Index RC.
Output: Updated RC-Index RC.

```
1  Insert(x, RC.root)
2  return RC
3  Function Insert(x, node)
4      Insert x into the cover tree node.CT
5      Update node.im based on x
6      if node.im ∉ [β, 1 − β] then
7          Batch construct the tree rooted at node with x included
8          return
9      foreach child ∈ {node.left, node.right, node.next} do
10         if x ∈ child.range then
11             Insert (x, child)
```

According to the master theorem, $t(n) = O(\gamma^6 n \log^{m+2} n)$. So the overall complexity is $T_{m+1}(n) = O(n \log n) + t(n) = O(\gamma^6 n \log^{m+2} n)$. □

3.4.2.2 Insertion and Deletion

Algorithm 10 lists the pseudocode of insertion. The deletion process is similar. Our idea is to apply a balance bound as in [121, 112]. Specifically, we define a rank of each node in our range tree as $rank(node) = (1 + \#\text{ nodes in subtree rooted at } node)$. A subtree here only means the subtree in the same dimension, excluding the nested subtrees in the next dimension. An empty tree has $rank = 1$ while a leave node has $rank = 2$. Then we define imbalance factor for each node as $im(node) = rank(node.left)/rank(node)$. Intuitively, an $im(node)$ closer to $1/2$ means a more balanced subtree rooted at $node$. A range tree after batch construction has $im \in [1/3, 2/3]$ for all nodes. In Algorithm 10, we insert a new item to the range tree and rebalance by batch constructing a subtree if its im falls out of $[\beta, 1-\beta]$, where $0 < \beta < 1/3$.

Our cover tree insertion, deletion, and querying algorithms are slightly different from the original one in the cover tree paper [15]. We must support general base distance $b > 1$ for our approach while the original algorithms only works when $b = 2$. The major dif-

Algorithm 11: Cover Tree Insertion

Input: Cover Tree CT; New item x
Output: Cover Tree CT with x inserted

Raise the level of $CT.root$ until it covers x
$x.parent \Leftarrow CT.root$
$x.level \Leftarrow CT.root.level - 1$
$factor \Leftarrow CT.b/(CT.b-1)$
$conflict \Leftarrow \{CT.root\}$
$level \Leftarrow CT.root.level$
$levelD \Leftarrow (CT.b)^{level}$
while $conflict \neq \emptyset$ **do**
    foreach $v \in conflict$ **do**
        if $dis(x,v) < factor \cdot levelD$ **then**
            if $dis(x,v) \leq levelD$ **then**
                $x.parent \Leftarrow v$
                $x.level \Leftarrow level - 1$
        else
            $conflict \Leftarrow conflict - \{v\}$
    $level \Leftarrow level - 1$
    $levelD \Leftarrow levelD/CT.b$
    $conflict \Leftarrow \{v | v.parent \in conflict \wedge v.level \geq level\}$

ference is that we need to compute an upper bound of the radius of a node. Suppose an item v is at level ℓ_v. Its direct children is within a ball of radius b^{ℓ_v}. Its "grand-children" is within a ball of radius $b^{\ell_v} + b^{\ell_v - 1}$ and so on. The limit of the radius is $\lim_{\ell \to \infty} b^{\ell_v}(1 - b^{-\ell})/(1 - b^{-1}) = b^{\ell_v} \cdot b/(b-1)$. So we precompute $factor = b/(b-1)$ as the upper bound factor for all items at different levels. An item x may conflict with an descendant of item v if $dis(x,v) < factor \cdot b^{\ell_v}$. Algorithm 11 shows the insertion procedure. Deletion and querying algorithms are very similar.

Theorem 3.4.9. *The amortized time complexity of insertion or deletion of RC-Index is* $O(\gamma^6 d \log^{d+2} n)$.

Our proof is similar to the proof in [112]. We define an imbalance score of the tree. An insertion or deletion may increase the imbalance score while a rebuild always decreases the imbalance score.

Proof. A pure insertion/deletion without batch construction visits $O(\log^d n)$ nodes and perform insertion/deletion on the corresponding cover trees. So the complexity is $O(\log^d n) \cdot O(\gamma^6 \log n) = O(\gamma^6 \log^{d+1} n)$.

Now we analyze the cost of batch construction. Let the imbalance score of a tree be

$$IM(tree) = \sum_{v_i \in tree} im(v_i) rank(v_i) \log^{d_i+1} rank(v_i)$$

where v_i are the nodes and d_i is the node's dimension. A node in the first dimension has $d_i = d$ and a node in the last dimension has $d_i = 1$.

Suppose we perform n operations each of which can be an insertion or deletion. So the tree has at most n items. Through an easy induction, we can prove that one insertion/deletion can increase $IM(tree)$ by $O(d \cdot \log^{d+2} n)$ (Lemma 3.4.10).

Given the imbalance score

$$IM(tree) = \sum_{v_i \in tree} im(v_i) rank(n_i) \log^{d_i+1} rank(v_i)$$

we have:

Lemma 3.4.10. *One insertion/deletion increases $IM(tree)$ by $O(d \cdot \log^{d+2} n)$.*

Proof. For each node n_i, an insertion/deletion can increase the imbalance factor $im(v_i)$ by at most $O(1/rank(v_i))$. So $im(v_i)rank(v_i)$ always increase by $O(1)$.

In the first dimension where $d_i = d$, we visit a path of $O(\log n)$ nodes. The sum of imbalance factor increments is thus $\sum_{\text{path}} O(1) \cdot \log^{d_i+1} rank(v_i) = \sum_{\text{path}} O(1) \log^{d+1} rank(v_i) \leq \sum_{\text{path}} O(1) \log^{d+1} n = O(\log^{d+2} n)$. In the second dimension where $d_i = d-1$, we visit $O(\log^2 n)$ nodes for all subtrees. The sum of imbalance factor increments is also $O(\log^{d+2} n)$. We have similar results for all dimensions.

So one insertion/deletion increases $IM(tree)$ by $O(d \cdot \log^{d+2} n)$ for all dimensions. □

So after n operations, $IM(tree)$ is at most $IM^+ = O(n \cdot d \cdot \log^{d+2} n)$.

In addition, each insertion/deletion can result in several batch constructions of subtrees. Assume u constructions occur during the process. Each construction reduces the imbalance factor im of a node from $(0,\beta) \cup (1-\beta,1)$ to $[1/3,2/3]$. In other words, it decreases im by at least $(1/3-\beta) = O(1)$. Formally, suppose every construction occurs at v_j in dimension d_j, it decreases im by at least $O(1) \cdot rank(v_j) \cdot \log^{d_j+1} rank(v_j)$. So all u constructions together decreases $IM(tree)$ by $IM^- = \sum_{j=1}^{u} O(1) \cdot rank(v_j) \cdot \log^{d_j+1} rank(v_j)$.

Initially, IM_0 is zero. After n operations, $IM_n = IM^+ - IM^-$ is positive. So $IM^- < IM^+$.

Based on Theorem 3.4.8, the cost of all constructions is $cost = \sum_{j=1}^{u} O(\gamma^6 rank(v_j) \cdot \log^{d_j+1} rank(v_j))$. So $cost = O(\gamma^6 \cdot IM^-) < O(\gamma^6 \cdot IM^+) = O(\gamma^6 \cdot n \cdot d \cdot \log^{d+2} n)$. So the amortized construction complexity of each operation is $O(\gamma^6 d \log^{d+2} n)$, dominating the pure insertion/deletion cost $O(\gamma^6 \log^{d+1} n)$. Therefore the amortized complexity of each operation is $O(\gamma^6 d \log^{d+2} n)$. □

3.4.2.3 Space Complexity

The actual space complexity of a cover tree on n items is $O(n)$, although the number of conceptual levels of a cover tree is infinite. We store only one record for each distinct item containing its id, highest level number, parent id, and children ids. So even though an item appears in many levels, we only store it once.

Theorem 3.4.11. *The RC-Index on n items takes $O(n \log^d n)$ space.*

Proof. We again prove it through induction.

• When $d = 1$, each node of the range tree is a cover tree. The root has n items, each of its two children has $n/2$ items, and so on. The space complexity of a cover tree is $O(n)$, so the 1-dimensional RC-Index takes $P_1(n) = O(n \log n)$ space.

• Suppose the space complexity is $O(n \log^m n)$ when $d = m$, we now prove the space is $O(n \log^{m+1} n)$ when $d = m+1$.

Remember we build $(m+1)$-dimensional RC-Index by splitting the first dimension and building an m-dimensional RC-Index for each node. So we have the following recurrence relation:

$$\begin{aligned} P_{m+1}(n) &= 2 \cdot P_{m+1}(n/2) + P_m(n) + O(n) \\ &= 2 \cdot P_{m+1}(n/2) + O(n \log^m n) + O(n) \\ &= 2 \cdot P_{m+1}(n/2) + O(n \log^m n) \end{aligned}$$

So $P_{m+1}(n) = O(n \log^{m+1} n)$ based on the master theorem. □

3.5 Index Selection

In this section, we briefly discuss how to build indexes given a set of queries. Before this section, we only consider building one specific RC-Index *RC* to answer a given query q. Formally, suppose a range query q applies filters on an attribute set $\mathbb{A}_q$ and the Range Index of *RC* is built on an attribute set $\mathbb{A}_{RC}$. We have only considered the case where $\mathbb{A}_{RC} = \mathbb{A}_q$. Now let's see some interesting properties of RC-Index when $\mathbb{A}_{RC}$ may not equal $\mathbb{A}_q$. These properties are useful when we discuss index selection.

Theorem 3.5.1. *The database can utilize RC to evaluate q with approximation ratio* $\frac{b-1-2b^{1-\delta}}{2(b-1)}$ *if* $\mathbb{A}_q \subseteq \mathbb{A}_{RC}$.

Proof. This is simply due to the property of the range tree. Recall how we traverse a range tree to answer a query q. Initially, q corresponds to a single search at the root of the range tree. If q applies a filter on the first dimension, we transform the root to $O(\log n)$ subordinate searches. However, if q applies no filter on the first dimension, we simply goes from the root to its child in the next dimension without branching new searches. Similar procedure happens in all dimensions. We still end up with multiple subordinate searches and extract items from the corresponding cover trees. Obviously, the approximate ratio in Theorem 3.4.4 still holds. So we can use *RC* to answer query q as long as $\mathbb{A}_q \subseteq \mathbb{A}_{RC}$. □

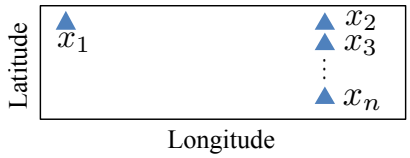


Figure 3.8: When $\mathbb{A}_q \supset \mathbb{A}_{RC}$, naïvely extend *RC* to answer q can result in arbitrarily bad diversity score.

Then a natural question is: can we utilize *RC* when $\mathbb{A}_q \supset \mathbb{A}_{RC}$? Specifically, can we use *RC* to find a diverse set of items satisfying filters on $\mathbb{A}_{RC}$ and then apply filters on $(\mathbb{A}_q - \mathbb{A}_{RC})$? Unfortunately, the output diversity score of this approach can be arbitrarily bad. For example, consider a special case where $\mathbb{A}_q = \mathbb{A}_{dis} = \{\texttt{Latitude}, \texttt{Longitude}\}$. The distance function is still Euclidean. $\mathbb{A}_{RC} = \{\texttt{Latitude}\} \subset \mathbb{A}_q$. Assume $k = 2$. As Figure 3.8 illustrates, *RC* may mistakenly pick diverse items within $\{x_2, ..., x_n\}$ because their latitudes are different from each other. However, the best diverse set should be $\{x_1, x_n\}$. In this case, the diversity score is arbitrarily bad because $dis(x_2, x_n)/dis(x_1, x_n)$ can approach zero.

Following the proof of Theorem 3.5.1, we can easily derive the time complexity of evaluating q. Assume $d = |\mathbb{A}_{RC}|$ and $d_q = |\mathbb{A}_q|$. We have:

Theorem 3.5.2. *The database can utilize RC to evaluate q with time complexity* $O(k^2 \gamma^{A(\delta+1)} \log^{d_q} n)$ *if* $\mathbb{A}_q \subseteq \mathbb{A}_{RC}$ *and* $d = O(k^2)$.

$d = O(k^2)$ means that the dimensionality of the RC-Index is not greater than order of k^2. This usually holds in practice. d is small because a user is unlikely to apply filters on more than 10 dimensions. But k could be easily greater than 5, resulting in $k^2 = 25$.

Proof. As we explain in the proof of Theorem 3.5.1, every time we apply a filter on a dimension, we branch $O(\log n)$ subordinate searches. So our algorithm ends up with $(\log^{d_q} n)$ subordinate searches in the range tree in $O((d - d_q) \log^{d_q} n)$ time where $(d - d_q)$ implies we go directly from one node to its child *next* in the next dimension without branching

new searches. Finally our algorithm extracts items from $O(\log^{d_q} n)$ cover trees to diversify them in $O(k^2\gamma^{A(\delta+1)}\log^{d_q} n)$ time. So the time complexity is $O\big((d-d_q)\log^{d_q} n + k^2\gamma^{A(\delta+1)}\log^{d_q} n\big)$. Since $(d-d_q) \leq d = O(k^2)$, the overall query time is dominated by $O(k^2\gamma^{A(\delta+1)}\log^{d_q} n)$. □

Given the above two properties of RC-Index, now we discuss how to build indexes given a set of queries $Q = \{q_1, q_2, ...\}$. Theorem 3.5.1 and 3.5.2 suggest that an RC-Index on $\mathbb{A}_{RC}$ can answer queries whose $\mathbb{A}_q \subseteq \mathbb{A}_{RC}$ with the same approximation ratio and time complexity as those when $\mathbb{A}_q = \mathbb{A}_{RC}$. So a naïve plan is to build a large index that covers all queryable attributes, i.e. $\mathbb{A}_{RC} = \cup_{q\in Q}\mathbb{A}_q$. The only problem of this plan is its large space complexity, $O(n\log^{|\mathbb{A}_{RC}|} n)$. We cannot build this large RC-Index when the database has a small space limit. Then the index selection becomes an optimization problem that optimizes the runtime and the approximation ratio of a workload while satisfying the space constraint. We defer the study of this problem in our future work.

3.6 MAXSUM

We show how to extend our framework to support MAXSUM in this section.

We build the same RC-Index on the items. Then we apply the same Algorithm 7 and 6 to extract candidates. Finally, we can change the greedy Algorithm 8 for our diversification module to a very similar greedy algorithm in [137] with $1/4$ approximation ratio to support MAXSUM. We prove the approximation ratio of this pipeline is $\frac{b-1-2b^{1-\delta}}{4(b-1)}$.

Assume the optimal solution is $S^* = \{x_1, x_2, ..., x_k\}$ with diversity score f^*. The candidate items we extract is X^C. Suppose we construct the same $Y = \{y_1, y_2, ..., y_k\}$ on X^C with diversity score f^Y as we do for Theorem 3.4.1. Let $K = \frac{k(k-1)}{2}$ where $k \geq 2$ for non-trivial cases. We have the following lemma and theorem.

Lemma 3.6.1. $dis(x_i, y_i) \leq \frac{b^{-\delta}}{1-b^{-1}}\frac{1}{K}\cdot f^*$.

Proof. Similarly, $dis(x_i, y_i)$ has two cases:

• Case 1: When the tree CT has no more than k items, since $y_i = x_i$, $dis(x_i, y_i) = 0 \leq \frac{b^{-\delta}}{1-b^{-1}} \frac{1}{K} \cdot f^*$.

• Case 2: When the tree CT has more than k items, any k items at Level ℓ_k also exist at Level $(\ell_k - \delta)$ due to **Nesting** and should be extracted. They have already formed a solution with diversity score b^{ℓ_k} because of **Separation**. In addition, as f^* is the optimal diversity score of all items, $b^{\ell_k} K \leq f^*$.

Since y_i is the ancestor of x_i and y_i is at Level $(\ell_k - \delta)$, according to **Covering**,

$$dis(x_i, y_i) < \sum_{\ell=-\infty}^{\ell_k - \delta} b^{\ell} = \frac{b^{\ell_k - \delta}}{1 - b^{-1}}$$

So

$$dis(x_i, y_i) < \frac{b^{\ell_k - \delta}}{1 - b^{-1}} = \frac{b^{-\delta}}{1 - b^{-1}} \cdot b^{\ell_k} \leq \frac{b^{-\delta}}{1 - b^{-1}} \cdot \frac{1}{K} f^*$$

□

Theorem 3.6.2. *Let the optimal diversity score on X be f^*, and the optimal diversity score on extracted X^C be f^C, $f^C \geq \frac{b-1-2b^{1-\delta}}{b-1} \cdot f^*$.*

Proof. According to triangle inequality and symmetry (Definition 3.2.3), for any x_i and the corresponding y_i:

$$dis(x_i, x_j) \leq dis(x_i, y_i) + dis(y_i, y_j) + dis(x_j, y_j)$$

equivalent to

$$dis(y_i, y_j) \geq dis(x_i, x_j) - dis(x_i, y_i) - dis(x_j, y_j)$$

Since f^* is the diversity score of $\{x_1, ..., x_k\}$, $\sum_{1 \leq i < j \leq k} dis(x_i, x_j) = f^*$. So for any distinct y_i and y_j:

$$
\begin{aligned}
\sum_{1\le i<j\le k} dis(y_i,y_j) &\ge \sum_{1\le i<j\le k} \big(dis(x_i,x_j) - dis(x_i,y_i) - dis(x_j,y_j)\big) \\
&= \sum_{1\le i<j\le k} dis(x_i,x_j) - \sum_{1\le i<j\le k} \big(dis(x_i,y_i) + dis(x_j,y_j)\big) \\
&\ge f^* - \sum_{1\le i<j\le k} \left(\frac{b^{-\delta}}{1-b^{-1}} \frac{1}{K} \cdot f^* + \frac{b^{-\delta}}{1-b^{-1}} \frac{1}{K} \cdot f^* \right) \\
&= f^* - K \left(2 \cdot \frac{b^{-\delta}}{1-b^{-1}} \frac{1}{K} \cdot f^* \right) \\
&= (1 - \frac{2b^{-\delta}}{1-b^{-1}}) \cdot f^* \\
&= \frac{b-1-2b^{1-\delta}}{b-1} \cdot f^*
\end{aligned}
$$

Since $f^C \ge f^Y \ge dis(y_i,y_j)$, $f^C \ge \frac{b-1-2b^{1-\delta}}{b-1} \cdot f^*$. □

After extracting the candidates, we apply a $1/4$-approximation algorithm for MAXSUM in [137]. So the overall approximation ratio is $\frac{b-1-2b^{1-\delta}}{4(b-1)}$.

3.7 Experimental Evaluation

In this section we evaluate our approach. Our results demonstrate that our approach can extract high quality diverse items efficiently.

3.7.1 Settings

3.7.1.1 Data

We experiment with two types of datasets. Figure 3.9 lists the real-world datasets. We obtain the City dataset containing $5,922$ Greek cities and villages from [46, 47]. Bank [119] and Census are two most popular datasets with $\ge 10,000$ items from UCI Machine Learning Repository [105]. Forest [16] and Gas [78] are two larger datasets from the same repository. We remove categorical attributes because their small domain over-simplifies the diversification problem. Then we randomly pick the remaining numerical attributes as $\mathbb{A}_q$ and $\mathbb{A}_{dis}$. We also generate synthetic datasets whose attributes follow uniform distributions.

Dataset	# Instances	Description
City	5,922	Greek cities
Bank	45,211	Marketing data of a Portuguese bank
Census	48,842	Census income of US
Forest	581,012	Forest cover type in Colorado, US
Gas	928,991	Gas sensors

Figure 3.9: Datasets.

We vary the following parameters one at a time in our experiments. The bold value of each parameter is the default value when we are not varying this parameter in any individual experiment. We vary n within $\{10^3, 5\times10^3, 10^4, \mathbf{5\times10^4}, ..., 10^6\}$ and k in $\{\mathbf{10}, 50, 100, 150, 200\}$. The query q applies filters on $\{\mathbf{1}, 2, 3\}$ attributes always covering 80% items. The distance function is Euclidean on another two attributes. We vary b in $\{1.1, 1.5, \mathbf{2.0}, 3.0, 4.0\}$ and δ in $\{0, 1, 2, \mathbf{3}, 4, 5\}$, which means $1/4$ approximation ratio when $b = 2.0$ and $\delta = 3$ by default. Some (b, δ) combinations makes the ratio less than 0, but they work well in practice as we will show in Section 3.7.3.

3.7.1.2 Configuration

We use a MacBook with 2GHz dual-core Intel Core i7 and SSD. Our prototype system runs in PostgreSQL, which utilizes 4GB memory.

We implement our prototype as user defined functions (UDFs) using C language at server side. The user can specify the attributes that allow range queries and also a distance function on certain attributes. Then we create two auxiliary tables as our indexes: one for Range Index and the other for Diversity Index. The user can conduct batch construction, insert items, or query diverse items by invoking our UDFs.

Such UDF-based implementation is flexible because it does not force a user to modify the source code of PostgreSQL. A user may not have the privilege to install a customized PostgreSQL if s/he is not the admin. GiST [73] also has such flexibility but there is no simple way to manipulate the inner nodes of an index tree using GiST. So we end up

Algorithm	Query Complexity	Approx. Ratio	Max Ratio
RC-Index	$O(k^2\gamma^{4(\delta+1)}\log^d n)$	$\frac{b-1-2b^{1-\delta}}{2(b-1)}$	$< 1/2$
Greedy	$O(kn)$	$1/2$	$1/2$
Tree	$O(n^2)$	$\frac{(b-1)}{2b^2}$	$\leq 1/8$
Tree++	$O(\gamma^6 n\log n)$	$\frac{(b-1)}{2b^2}$	$\leq 1/8$

Figure 3.10: Algorithms.

with UDFs. We can adapt the UDFs to seamless triggers in the future to improve user experience. The efficiency of our UDFs is not an issue as we will demonstrate soon in this section.

3.7.1.3 Comparison with Baselines

We compare with three baseline approaches. They are the state-of-the-art to the best of our knowledge.

• `Greedy`. Greedy algorithm selects a random item to initialize the diverse result set S. Then for every unselected item x, it maintains an array of the minimum distance between x and any item in S. It iteratively adds the item with the maximum minimum distance into S and updates the distance array. Its pseudo code is the same as Algorithm 8 but it takes all items in $q(X)$ as input. So in the worst case, its query time is $O(kn)$. This is a $1/2$-approximation algorithm [147, 137].

• `Tree`. Drosou and Pitoura [47] have developed an approximation algorithm based on a single cover tree. They assume the items continuously get into the system so they perform insertion and deletion while selecting diverse items from the cover tree. We adapt this approach to support arbitrary range queries by updating the cover tree between queries. Their streaming data scenario is *different* from ours so such an adaptation is slower than our approach. Its batch construction time for building a cover tree is $O(n^2)$. In addition, its maximum approximation upper bound is $1/8$ when $b = 2.0$. But our approach can achieve, for example, $1/4$ approximation ratio within very short query time.

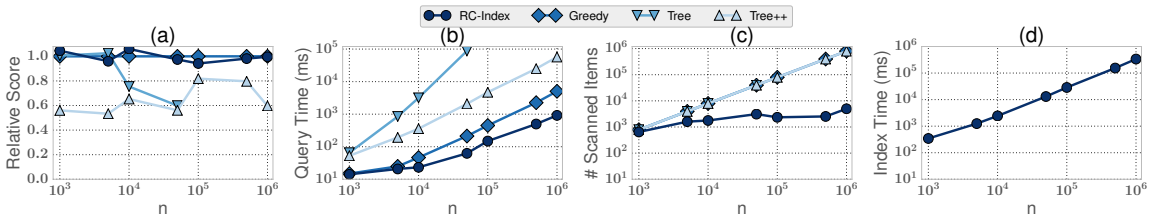


Figure 3.11: Synthetic data with varying size n: `RC-Index` is nearly an order of magnitude faster than the state-of-the-art while ensuring high quality result.

• `Tree++`. This is the original cover tree algorithm [15], which can be viewed as a variant of the above approach. Instead of finding the farthest pair of items to construct a tree in `Tree`, we randomly pick an item to start the tree construction in `Tree++`, which reduces $O(n^2)$ complexity to $O(\gamma^6 n \log n)$.

Figure 3.10 shows the worst-case query complexity and approximation ratio of all algorithms. At a glance, all three baselines' query complexity is $\Omega(n)$ because they have to scan the entire $q(X)$ when $|q(X)| = O(n)$ in the worst case. But our complexity has no $O(n)$ term with the help of indexes. Our approximation ratio $\alpha = (b-1-2b^{1-\delta})/2(b-1)$ is better than `Tree` and `Tree++`'s. Moreover, α's upper limit equals `Greedy`'s ratio $1/2$, which is the best possible approximation ratio of a polynomial algorithm unless P=NP [137].

Since `Greedy` has the best approximation ratio while finding the optimum solution takes exponential time, we compare the diversity scores against `Greedy`'s. We compute the *relative score* for each algorithm *Algo* as $f_{Algo}/f_{\texttt{Greedy}}$ in the following experiments.

3.7.2 Quality and Scalability

We vary n and k to see how diversity score and runtime change. We also show the number of scanned items $|X^C|$ to help demonstrate that our approach is more efficient because we scan much fewer items.

Figure 3.11 compares the performance on synthetic data where n varies within $\{10^3, 5\times 10^3, 10^4, ..., 10^6\}$. We do not run `Tree` on $n \geq 10^5$ because its query time is too long. (1)

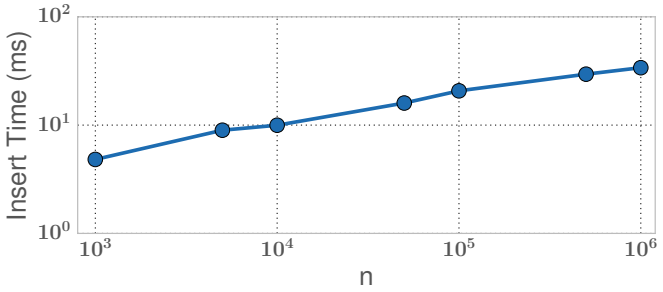


Figure 3.12: Insertion time of individual item increases as n grows.

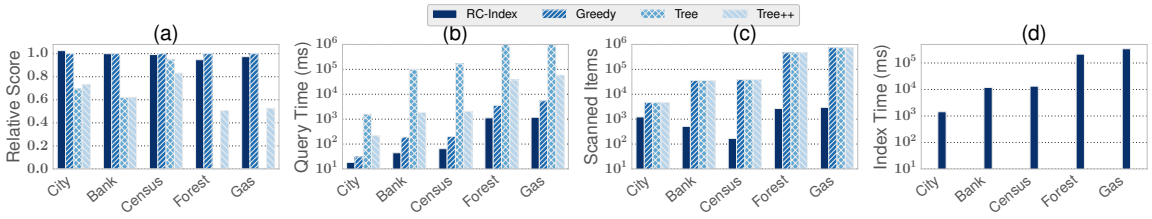
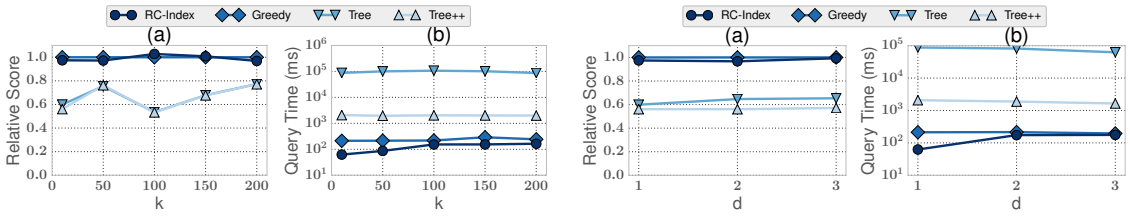


Figure 3.13: Real-world data: `RC-Index` is the fastest approach while ensuring high quality result.

Figure 3.11a depicts the relative score compared to `Greedy`'s. The quality of our result is as good as `Greedy`'s, and they are both better than `Tree` and `Tree++` when $n \geq 10^4$. In some cases, `RC-Index` outperforms `Greedy` even though its theoretical approximation ratio is worse than `Greedy`'s. This is simply because the synthetic data is not the worst case data for `RC-Index`. (2) Figure 3.11b shows that `RC-Index` is an order of magnitude faster than the state-of-the-art, `Greedy` and `Tree++`. Our query time is 0.9 second when there are 10^6 items. (3) Figure 3.11c tells us why `RC-Index` is efficient: it scans far fewer candidate items than the other baselines do. (4) Finally, Figure 3.11d shows the batch index creation time of the synthetic data. It is a *one-time* cost after loading the data. `RC-Index` can index 10^5 items within only 28 seconds, or 10^6 items within 330 seconds. The indexing time increases almost linearly with regard to n (both in log scale in the figure), which is a desired property.



(a) (b)

Figure 3.14: (a) Varying k: `RC-Index` delivers good result in short query time when k increases. (b) Varying d: `RC-Index`'s query time gets closer to Greedy's but the result quality is the same as d increases.

We also plot the insertion time as n grows. For each n in $\{10^3, 5\times10^3, 10^4, ..., 10^6\}$, we insert 100 new items into RC-Index and compute the average insertion time. As Figure 3.12 illustrates, the insertion time grows slowly as n increases.

RC-Index also outperforms the others on real-world data as illustrated in Figure 3.13. We do not run `Tree` on Forest and Gas because its query time is too long. Figure 3.13a shows that `RC-Index`'s score is as good as `Greedy`'s and is better than `Tree` and `Tree++`. Figure 3.13b shows our query time is orders of magnitude less than the other three baselines, because we scan much fewer items as depicted in Figure 3.13c. Finally, Figure 3.13 shows our indexing time is short: only 13.3 seconds for $48,842$ items or 343.8 seconds for $928,991$.

Now we fix $n = 5\times10^4$ and vary k in $\{10, 50, 100, 150, 200\}$. As we can see in Figure 3.14aa, the quality of result is roughly the same as k increases. Figure 3.14ab also shows the query time does not increase much for larger k.

Next, we vary d in $\{1,2,3\}$ while fixing $n = 5\times10^4$ and $k = 10$. Figure 3.14ba demonstrates that the quality is stable. Figure 3.14bb shows the query time increases because the query time complexity is proportional to $\log^d n$.

Finally, we change the distance function from Euclidean to Manhattan distance and compare all algorithms as n grows. Figure 3.15 depicts the relative score and query time.

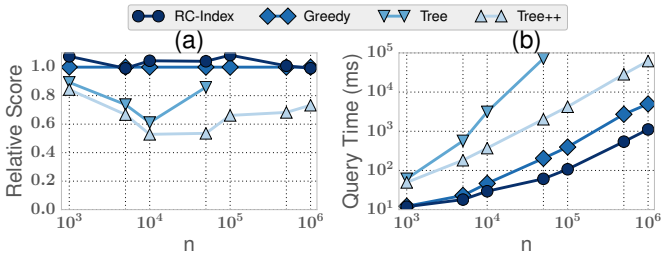
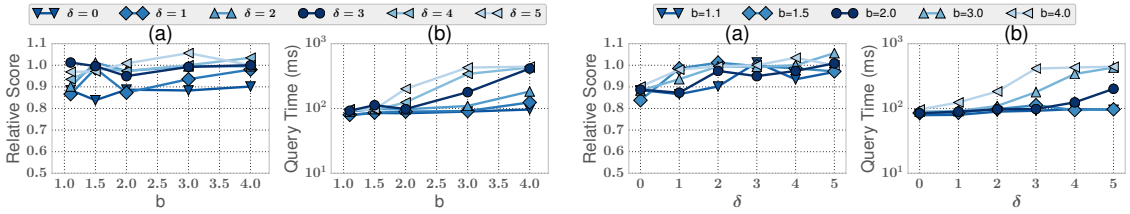


Figure 3.15: Comparison under Manhattan distance. `RC-Index` outperforms the other algorithms.



(a) (b)

Figure 3.16: (a) Varying b: greater b means longer query time. (b) Varying δ: greater δ means better result but longer runtime.

Similar to the performance under Euclidean distance, `RC-Index` is nearly an order of magnitude faster than the other algorithms and provides high diversity score.

3.7.3 Sensitivity

We test the sensitivity of our approach on synthetic data with 10^5 items in this section. Two parameters, b and δ, impact our approximation ratio $\frac{b-1-2b^{1-\delta}}{2(b-1)}$ and query complexity $O(k^2\gamma^{A(\delta+1)}\log^d n)$. We vary them to see how performance changes. Note that some (b,δ) combinations makes $\frac{b-1-2b^{1-\delta}}{2(b-1)} \leq 0$, but they actually lead to good result in practice. So we still present them in the following figures.

Figure 3.16a depicts the performance using different b. We vary b in $\{1.1, 1.5, 2.0, 3.0, 4.0\}$ for six different δ between 0 and 5. On the one hand, greater b means greater approximation ratio in the worst case. But we do not observe such trend in Figure 3.16aa, because the

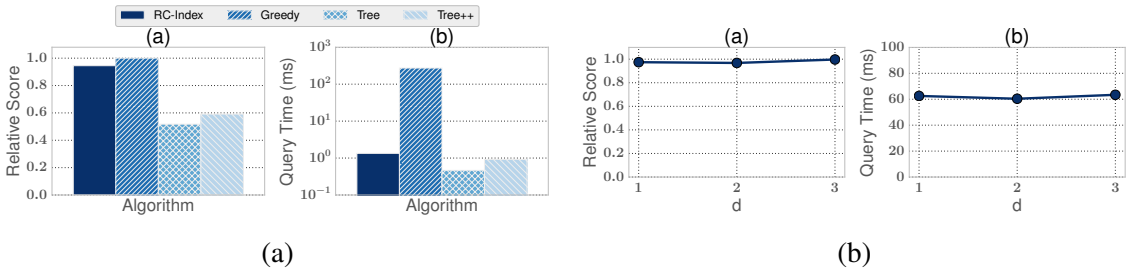


Figure 3.17: (a) Our algorithm works well on streaming data. (b) RC-Index supports partial range query well.

synthetic data following uniform distribution is not the worst-case data. It is unusual to run into worst-case scenario in practice. On the other hand, greater b means greater expansion constant γ. So the query time is longer as Figure 3.16ab shows. When b and δ are large enough, the query time converges because the algorithm has extracted all items from $q(X)$.

Figure 3.16b shows how performance changes along with δ. We vary δ in $\{0,1,2,3,4,5\}$ but fix b at different values this time. Intuitively, greater δ means our algorithm goes deeper in each cover tree to extract more candidate items. So the diversity score gets slightly better in general as Figure 3.16ba depicts. The cost is that the query time gets longer as in Figure 3.16bb. Again, the query time converges when b and δ are large, because the algorithm has extracted the entire $q(X)$.

3.7.4 Data Stream

In this experiment, we examine the performance of RC-Index on streaming data. We redesign and reimplement the algorithms in Figure 3.10 and apply them on a data stream. We query this data stream through a sliding window. The window covers $50,000$ items. Every time we slide the window, we remove 10 old items, add 10 new items, and query the diverse items again.

We compare the performance of four algorithms on 10 queries and plot the average relative score and query time in Figure 3.17a. The diversity score of our algorithm is close

to the best score and better than `Tree` and `Tree++`. In terms of query time, `Tree` [47] is the fastest approach, but our algorithm is only slightly slower than `Tree`.

3.7.5 Index Selection

In this experiment we demonstrate Theorem 3.5.1 and 3.5.2. Given a range query q that applies filters on a set of attributes $\mathbb{A}_q$, one can build an RC-Index whose Range Index is on an attribute set $\mathbb{A}_{RC}$ to answer q as long as $\mathbb{A}_q \subseteq \mathbb{A}_{RC}$. Theorem 3.5.1 and 3.5.2 show that the diversity score and query time are irrelevant to the size of $\mathbb{A}_{RC}$. Here we fix $|\mathbb{A}_q| = 1$ and vary $d = |\mathbb{A}_{RC}|$ in $\{1,2,3\}$. As Figure 3.17b depicts, the diversity score and query time are stable as d grows.

3.8 Related Work

Search result diversification is about selecting a small subset of diverse items to present to the user when a large set of items satisfy the user query. It finds its application in many scenarios such as data exploration, Web search, and recommendation systems. According to the survey [45, 139, 186], diversification problems can be classified into three categories: (1) content-based (or similarity-based) diversification finds items that are dissimilar to each other; (2) intent-based (or semantic coverage-based) diversification finds items relevant to various topics to help user further disambiguate the query; (3) novelty-based diversification finds items that are different from the previously retrieved ones for the user. Our work focuses on content-based diversification, which mainly maximizes the distance between selected items and the relevance of items to the query [60, 42, 43]. In our setting, we treat relevance score of an item as either relevant or irrelevant rather than a continuous score because we are dealing with conventional relational queries. We select only relevant items to present to the user.

There are two most common objective functions in content-based diversification problems: MAXMIN and MAXSUM. MAXMIN maximizes the minimum pairwise distances

between selected items while MAXSUM maximizes the sum of pairwise distances. Both problems have been studied earlier in operational research as dispersion problems. The original motivation is to locate undesired facilities like nuclear reactors among the given nodes in a network. These two problems are proved to be NP-hard in discrete and continuous cases [51, 147, 137]. We focus on MAXMIN as its result is more representative in many applications like geolocation based diversifications. Ravi et al. [137] prove that MAXMIN is not only NP-hard but also APX-hard for general distance functions. But when the distance function obeys the triangle inequality, a greedy heuristic results in a $1/2$-approximation algorithm, and no polynomial algorithm can achieve better performance guarantee unless P=NP.

However, researchers cannot directly migrate this greedy $1/2$-approximation algorithm to a database because it is too expensive. The time complexity of this algorithm is $O(kn)$ where n is the number of items in a query result. A database can have a million items satisfying a query in a data exploration or product search scenario. So $O(kn)$ can be very large even when k is as small as ten. Its bottleneck is the scan of $O(n)$ items. Yu et al. [183] have developed a similar approach which starts with k items and swaps better items with them greedily. Carbonell and Goldstein [24] iteratively selects items with maximal marginal relevance. Vieira et al. [158] merge more scores and apply randomization in the greedy algorithm. Khan et al. [89] classify the above techniques as "process-first-diversify-next", which are expensive because they may scan or sort $O(n)$ items in the worst case. Our index-based approach avoids this issue to be much faster.

Many researchers use indexes to improve the efficiency. Vee et al. [157] work on categorical distance and therefore use index to probe diverse items. Qin et al. [129] considers binary distance. So they reduce the problem to weighted independent set and solve it with the help of the graph structure. But our approach are dealing with more general distance functions. Some researchers reuse the selected items to further shorten query evaluation time. Several papers [118, 47, 44, 25] work on extracting diverse items from continuous

data stream. While new items getting into the system, user also continuously query the stream. So they can reuse processed old items as result. Some other papers [90, 91, 92] assume correlation between consecutive queries during data exploration. So they can reuse previously returned items. Our approach does not make such assumptions. We support any range queries efficiently with quality guarantee, making our approach applicable to more general scenarios. Drosou and Pitoura [46] also utilize index, but they solve a variant of diversification problem where the distance between selected items only need to be greater than a given threshold. Khan and Sharaf [89] prune items according to the sorted partial distance. But they focus on MAXSUM and the performance guarantee is unclear. We focus on MAXMIN and our algorithm has an approximation bound.

Our approach uses cover tree [15], which is originally designed for nearest neighbor search. Some other early data structures like ball tree [123], metric skip list [88], and navigating net [97] also have similar features. We use cover tree as our Diversity Index because of its simplicity and good runtime performance in practice, but our approach does not stick to any particular data structure.

The study of range query dates back to 1970s. Researchers propose k-d tree [11], quad tree [53], B+ tree [36], VA-file [164], range tree [111, 12, 166, 100], and so on. We use range tree because of its good time complexity [166, 100, 112]. The range tree in our approach can be replaced by any other range query index if necessary.

3.9 Discussion

Currently, we make no assumption about the workload. So we build balanced Range Index (RI) to answer each range query with $O(\log^d n)$ subordinate searches. However, in practice, some ranges may be very popular, so a system designer can directly specify the bounds of the inner nodes of an RI. For example, if many users search for laptops between $\$1,000$ and $\$3,000$, the system designer can force an inner node of RI to cover $[\$1,000, \$3,001)$. Remember, our framework does not rely on any particular RI. We can

build cover trees on any partition. The problem then becomes another optimization problem whose target is minimize the query cost and index construction cost to answer a set of queries.

CHAPTER 4

A CONSUMER-CENTRIC MARKET FOR DATABASE COMPUTATION IN THE CLOUD

4.1 Introduction

The availability of public computing resources in the cloud has revolutionized data analysis. Users no longer need to purchase and maintain dedicated hardware to perform large-scale computing tasks. Instead, they can execute their tasks in the cloud with the appealing opportunity to pay for just what they need. They can choose virtual machines with a wide variety of computational capabilities, they can easily form large clusters of virtual machines to parallelize their tasks, and they can use software that is already installed and configured.

Yet, taking advantage of this newly-available computing infrastructure often requires significant expertise. The common pricing mechanism of the public cloud requires that users think about low-level resources (e.g. memory, number of cores, CPU speed, IO rates) and how those resources will translate into efficiency of the user's task. Ultimately, users with a well-defined computational task in mind care most about two key factors: the task's completion time and its financial cost. Unfortunately, many users lack the sophistication to navigate the complex options available in the cloud and to choose a configuration[1] that meets their preferences.

As a simple example, imagine users who need to execute a workload of relational queries using the Amazon Relational Database Service (RDS). They need to select a ma-

[1] A configuration here means a set of system resources and its settings, provided by the cloud provider. It includes the number of virtual instances of a cluster, the buffer size of a cloud database, and so on.

chine type from a list of more than 20 possible options, including "db.m3.xlarge" (4 virtual CPUs, 15GB of memory, costing $0.370 per hour) and "db.r3.xlarge" (4 virtual CPUs, 30.5GB of memory, costing $0.475 per hour). The query workload may run more quickly using db.r3.xlarge, because it has more memory, however the hourly rate of db.r3.xlarge is also more expensive, which may result in higher overall cost. Which machine type should the users choose if they are interested in the cheapest execution? Which machine type should they choose if they are interested in the cheapest execution completing within 10 minutes? Typical users do not have enough information to make this choice, as they are often not familiar with configuration parameters or cost models.

The reality of users' choices is even more complex since they may choose one of five data management systems through RDS, or other query engines using EC2, including parallel processing engines, and different configuration options for each. They might also be tempted to compare multiple service providers, in which case they would have to deal with different pricing mechanisms in addition to different configuration options. Amazon RDS charges based on the capacity and number of computational nodes per hour; Google BigQuery charges based on the size of data processed; Microsoft Azure SQL Database charges based on the capacities of service tiers like database size limit and transaction rate.

As a result of this complexity, many users of public cloud resources make naïve, suboptimal choices that result in overpayment, and/or performance that is contrary to their preferences (e.g., it exceeds their desired deadline or exceeds their budget). Thus, instead of paying only for what they need, the reality is that they pay for what they do not need and, even worse, they pay more than they have to for it.

A market for database computations: To ease the burden on users we propose a new market-based framework for pricing computational tasks in the cloud. Our framework introduces an entity called an *agent*, who acts as a broker between consumers and cloud service providers. The agent accepts data and computational tasks from users, estimates

the time and cost for evaluating the tasks, and returns to consumers *contracts* that specify the price and completion time for each task.

Our market can operate in conjunction with existing cloud markets, as it does not alter the way cloud providers offer and price services. It simplifies cloud use for consumers by allowing them to compare contracts, rather than choose resources directly. The market also allows users to extract more value from public cloud resources, achieving cheaper and faster query processing than naive configurations. At the same time, a portion of the value an agent helps extract from the cloud will be earned by the agent as profit.

Agents are conceptually distinct from cloud service providers in the sense that they have their intelligent models to estimate time and cost given consumers queries. In other words, agents take the risk of estimation, while service providers simply charge based on resource consumption, which guarantees profit. In practice, an agent could be a service provider (who provides estimation as a service in addition to cloud resources), a piece of software sold to consumers, or a separate third party who provides service across multiple providers.

Scope: Our goal in this chapter is *not* to develop a new technical approach for estimating completion time or deriving an optimal configuration for a cloud-based computation. Prior work has considered these challenges, but, in our view, has not provided a suitable solution to the complexity of cloud provisioning. The reason is that estimation, even for relatively well-defined tasks like relational workloads, is difficult. Proposed methods require complicated profiling tasks to generate models and specialize to one type of workload (e.g., Relational database [87] or MapReduce [74]). In addition, there is inherent uncertainty in prediction, caused by multi-tenancy common in the cloud [160, 140, 52, 152, 93]. Lastly, users' preferences are complex, involving both completion time [134] and cost [128, 181, 95, 187, 104], which have been considered as separate goals [75, 113, 115], but have not been successfully integrated.

Our market-based framework incentivizes expert agents to employ combinations of existing estimation techniques to provide this functionality as a service to non-expert consumers. Users can express preferences in terms of their *utility*, which includes both time and cost considerations. Uncertainty in prediction becomes a risk managed by agents, and included in the price of contracts, rather than a problem for users. Ultimately our work complements research into better cost estimation in the cloud [172, 30, 75]. In fact, our market will function more effectively as such research advances and agents can exploit new techniques for better estimation.

Our work makes several contributions:

- We define a novel market for database computations, including flexible contracts reflecting user preferences.
- We formalize the agent's task of pricing contracts and propose an efficient algorithm for optimizing contracts.
- We perform extensive evaluation on Amazon's public cloud, using benchmark queries and real-world scientific workflows. We show that our market is practical and effective, and satisfies key properties ensuring that both consumers and agents benefit from the market.

The chapter is organized as follows. We present an overview of the market and main actors in Section 4.2. We formally define contracts and optimal pricing of contracts in Section 4.3 and 4.4. We extend our framework to support fine-grained pricing to further optimize contracts in Section 4.5. In Section 4.6, we introduce several alternatives. In Section 4.7, we present a thorough evaluation of our proposed market, and demonstrate that it guarantees several important properties. Finally, we discuss related work and extension in Sections 4.8 and 4.9.

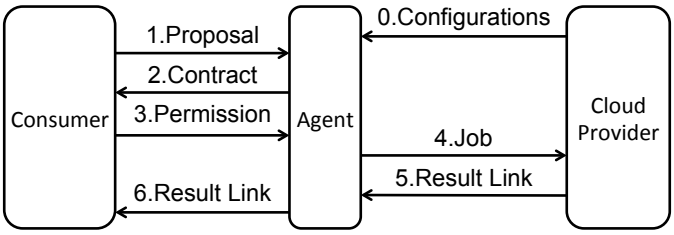


Figure 4.1: An overview of interactions of the main participants in the computation market: the consumer, the agent, and the cloud provider.

4.2 Computation Market Overview

In this section, we discuss the high-level architectural components of our computation market: three types of participants and their interactions through computation contracts. Our computation market exhibits several desirable properties, which we mention in Section 4.2.3.

4.2.1 Market Participants

Our goal is to model the interactions that occur in a computation market, and design the roles and framework in a way that ensures that the market functions effectively. Our computation market involves three types of participants:

- **Cloud provider.** Cloud providers are public entities that offer computational resources as a service, on a pay-as-you-go basis. These resources are often presented as virtual machine types and providers charge fees based on the capabilities of the virtual machines and the duration of their use. Our framework does not enforce any assumptions on the types, quantity, or quality of resources that a cloud provider offers.

- **Consumer.** A consumer is a participant in our computational market who needs to complete a computational task over a dataset D. We assume the computational task is a set of queries or MapReduce jobs[2], denoted as $\mathbf{Q} = \{Q_1, Q_2, ..., Q_n\}$. We assume that the con-

[2]For simplicity of terminology we use "query" to refer to either a query or MapReduce job.

sumer does not own the computational resources needed to complete **Q**, and thus needs to use cloud resources. However, the consumer may not have the expertise to determine which cloud provider to use, which resources to lease, or how to configure them. In our framework, the consumer wishes to retrieve the task results $\mathbf{Q}(D) = \{Q_1(D), Q_2(D), ..., Q_n(D)\}$ within a specified timeframe, and pay for these results directly. Therefore, the consumer's goal is to complete the task efficiently and for low cost. Different consumers have different time and cost preferences. They will describe these preferences precisely using a utility function, as described later in Section 4.3.1.

- **Agent.** Consumers' needs are task-centric (time and price to complete a given task), whereas cloud providers' abilities are resource-centric (time and price for a type of resource). Due to this disparity, consumers and providers do not interact directly in our framework. Rather, a semantically separate entity, the *agent*, is tasked with handling the interactions between consumers and cloud providers. The agent receives a task request from a consumer and, in response, calculates a price to complete the task, providing the consumer with a formal contract. We review contracts in Section 4.2.2, and describe them in detail in Section 4.3. The agent executes accepted contracts using public cloud resources, and earns a profit whenever the contract price is greater than the actual cost of executing the task. The agent's goals are to attract business by pricing contracts competitively and to earn a profit with each transaction. One of the main challenges for the agent is to assign accurate prices to consumer requests, which requires knowledge of cloud resources, their capabilities and costs, and expertise in tuning and query prediction.

Figure 4.1 illustrates the interactions among the three market participants. In step 0, the agent collects details on available configurations from the cloud provider to derive later price quotes on consumers' requests. This step may only need to be initiated once, and reused afterwards. In steps 1 through 3, the agent receives a proposal including **Q** and statistics about dataset D, denoted s_D, which are sufficient for pricing. For example, s_D can be the number of input records in each table [6], histograms on key columns or sets of

columns [170, 169, 171], a small sample of data [74], and other standard statistics relevant to the task. The agent reasons about possible configurations and estimates the completion time and financial cost of the queries, returning a priced contract to the consumer. If the consumer accepts the contract, in steps 4 through 6, the agent executes a job in the cloud according to the contract, computes the result, and returns a link to the consumer. The link can be, for example, an URL pointing to Amazon S3 or any other cloud storage service. Finally the agent receives payment based on the accepted contract and the actual completion time. We will see in Section 4.3.2 that contracts can involve complex prices that depend on the actual completion time.

4.2.2 Contracts

The contract is the core component of our framework, describing the terms of a computational task the agents will perform and the price they will receive upon completion of the task. The design of our market framework is intended to cope with the inevitable uncertainty of completion time. Therefore, our contracts support variable pricing based on the actual completion time when the answer is delivered.

We also formally model the time/cost preferences of the consumer using a *utility function* that we assume is shared with the agent. The main technical challenge for the agent is to price a contract of interest to a consumer. Pricing relies on the agent's model of expected completion time for the task as well as the consumer's utility. From the consumers' side, they may receive and compare contracts from multiple agents in order to choose the one that maximizes their utility.

In this chapter, we consider contracts and computational tasks that only involve analytic workloads. These analytic workloads are different from long-running services in the sense that their evaluation takes limited amount of time, even though this time can be several hours or days. Given this focus, we can assume that cloud resources do not change during the execution of task. This means, for example, that the capacity of virtual machines and

their rate remain the same during the execution of a contract. We discuss relaxing these factors in Section 4.9.

4.2.3 Properties and Assumptions

Our framework is designed to support three important properties: competitiveness, fairness, and resilience. *Competitiveness* guarantees that agents have an incentive to reduce runtime and/or cost for consumers. *Fairness* guarantees that agents have an incentive to present accurate estimates to consumers, and that they do not benefit by lying about expected completion times. *Resilience* means that an agent can profit in the marketplace even when their estimates of completion time are imprecise and possibly erroneous. We demonstrate empirically in Section 4.7 that our framework satisfies these crucial properties.

Our framework assumes honest participants; we defer the study of malicious consumers and agents to future work. Accepting an agent's contract means the consumer's data will be shared with the agent for evaluation of their task, however requesting contract prices from a set of agents reveals only the consumer's statistics and task description.

Monopoly is not possible in this framework, and collusion among agents is unlikely.[3] First, an agent cannot constitute a monopoly, since consumers may always choose to use a cloud service provider directly. A service provider cannot constitute a monopoly either, as any agent with a valid estimation model can enter the market. Second, collusion becomes unlikely as the number of agents in the market increases. Any agent who does not collude with others can offer a lower price and draw consumers, making any collusion unstable.

4.3 The Consumer's Point-of-View

In this section, we describe the consumer's interactions with the market. A transaction begins with a consumer who submits a request. This request reflects their *utility*, which

[3] In fact, the agents and the existing cloud service providers naturally form a monopolistic competition [155, 114].

is a precise description of their preferences. Later, given multiple priced contracts, the consumers can formally evaluate them according to the likely utility they will offer.

4.3.1 Consumer Utility

One of our goals is to avoid simplistic definitions of contracts in which a task is carried out by a deadline for a single price. For one, many consumers have preferences far more complex than individual deadlines: they can tolerate a range of completion times, assuming they are priced appropriately. In addition, we want agents to compete to offer contracts that best meet the preferences of consumers.

A consumer's preferences are somewhat complex because they involve tradeoffs between both completion time and price. We adopt the standard economic notion of consumer *utility* [155] and model it explicitly in our framework. A utility function precisely describes a consumer's preferences by associating a utility value with every (time, price) pair. A utility function can encode, e.g., the fact that the consumer is indifferent to receiving their query answer in 10 minutes at a cost of $2.30 or 20 minutes at a cost of $1.90 (when these two cases have equal utility values) or that receiving an answer in 30 minutes at a cost of $0.75 is preferable to both of the above (when it has strictly greater utility).

Definition 4.3.1 (Utility)**.** Utility $U(t,\pi)$ *is a real-valued function of time and price, which measures consumer satisfaction when a task is evaluated in time* t *with price* π*.*

Larger values for $U(t,\pi)$ mean greater utility and a preferred setting of t and π. For a fixed completion time t_0, a consumer always prefers a lower price, so $U(t_0,\pi)$ increases as π decreases. Similarly, for a fixed price, π_0, a consumer always prefers a shorter completion time, so $U(t,\pi_0)$ increases with decreasing t.

To simplify the representation of a consumer's utility, we will restrict our attention to utility functions that are piecewise linear. That is, we assume the range of completion times $[0,\infty)$ is divided into a fixed set of intervals, and that utility on each interval is defined by a

linear function of t and π. This means that for each interval, the consumer has a (potentially different) rate at which she/he is willing to trade more time for lower price, and vice versa.

Definition 4.3.2 (Utility – piecewise)**.** *A piece-wise utility function consists of a list of target times* $\tau_0, \ldots, \tau_n$*, where* $0 = \tau_0 < \tau_1 < \cdots < \tau_{n-1} < \tau_n = \infty$*, and linear functions* $u_1(\pi,t), \ldots, u_n(\pi,t)$*. The utility is* $u_i(\pi,t)$ *for* $t \in [\tau_{i-1}, \tau_i)$*.*

Such utility functions can express conventional deadlines, but also much more subtle preferences concerning the completion time and price of a computation.

Example 4.3.3. *Consumer Carol has two target completion times for her computation: 10 minutes and 20 minutes. Results returned in less than 10 minutes are welcome, but she doesn't wish to pay more to speed up the task. When results are returned between 10 minutes and 20 minutes, every minute saved is worth 1 cent to her. She does not want result returned after 20 minutes. Her piecewise utility function is:*

$$U(t,\pi) = \begin{cases} u_1(t,\pi) = -\pi & (t < 10) \\ u_2(t,\pi) = -t - \pi + 10 & (10 \le t < 20) \\ u_3(t,\pi) = -50 & (t \ge 20) \end{cases}$$

Figure 4.2 depicts $U(t,\pi)$ *when* $t < 20$*.*

In practice, users can construct the utility function by defining several critical points on a graphical user interface, or answering a few simple pair-wise preference questions.

4.3.2 Consumer Contract Proposal

The process of agreeing on a contract starts with the consumer advertising to agents the basic terms of a contract: the task $\mathbf{Q}$, the statistics of the database s_D, and their piecewise utility function U.

The terms of the contract are structured around the target times in the utility function. Agents use the utility function to choose a suitable configuration and pricing to match the

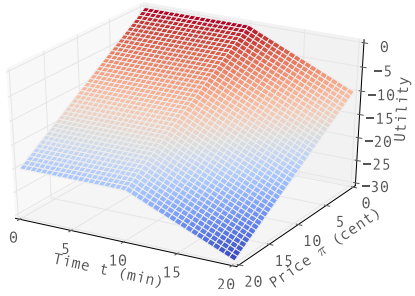


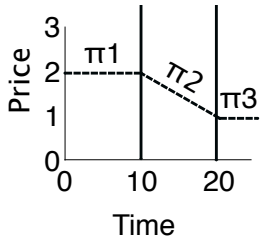
Figure 4.2: Utility function for Example 4.3.3 when $t < 20$.

preferences of the consumer. A complete, priced contract is returned to the consumer, which is defined as follows:

Definition 4.3.4 (Contract)**.** *A contract is a six-tuple* $\mathscr{C} = (\mathbf{Q}, s_D, \mathscr{T}, P, \hat{T}, \Pi)$*, where* $\mathbf{Q}$ *is a task,* s_D *consists of statistics about the input data,* $\mathscr{T} = (\tau_0, \tau_1, \ldots, \tau_n)$ *is an ordered list of target completion times,* $P = (p_1, \ldots, p_n)$ *is an ordered list of probabilities,* $\sum_i p_i = 1$*,* $\hat{T} = (\hat{t}_1, \ldots, \hat{t}_n)$ *is an ordered list of expected completion times, and* $\Pi = (\pi_1(t), \ldots, \pi_n(t))$ *is a list of price functions where* π_i *is defined on* $[\tau_{i-1}, \tau_i)$*.*

When a consumer and agent agree on a contract $\mathscr{C}$, it means that the agent has promised to deliver the answer to task $\mathbf{Q}$ on D after time $t \in [0, \infty)$, where the likelihood that t falls in interval $[\tau_{i-1}, \tau_i)$ is p_i. Accordingly, if the answer is delivered in the time interval $[\tau_{i-1}, \tau_i)$ the consumer agrees to pay the specified price, $\pi_i(t)$. $\hat{T}$ is used for computing expected utility as we will see in Section 4.3.3. The data statistics s_D are given to the agent by the consumer; the agent includes them in the contract because the pricing calculation relies on these statistics.

The contract is an agreement to run the task once. The probabilities provided by the agent are a claim that if the task were run many times, a fraction of roughly p_i of the time, the completion time would be in the interval $[\tau_{i-1}, \tau_i)$. Without this information, the consumer has no way to effectively evaluate the alternative completion times that could occur in a contract. For example, all alternatives but one could be very unlikely and this



(a)

Contracts	$\mathscr{C}_1$	$\mathscr{C}_2$
$\hat{T}$	$(9,15,21)$	$(9,15,21)$
P	$(0.2,0.5,0.3)$	$(0.1,0.8,0.1)$
Utility	-18.65	-10.4

(b)

Figure 4.3: (a) Price function for Example 4.3.5. (b) Comparison of two contracts.

would change the meaning of the contract. We will see in Section 4.4 how the agent generates these probabilities.

Example 4.3.5. *An example contract based on the utility function of Example 4.3.3 is defined by* $\mathscr{T} = (0,10,20,\infty)$*, probabilities* $P = (0.2,0.5,0.3)$*, expectations* $\hat{T} = (9,15,21)$*, and prices* Π *(also illustrated in Figure 4.3a) defined as:*

$$\Pi(t) = \begin{cases} \pi_1(t) = 2 & (t < 10) \\ \pi_2(t) = 3 - 0.1t & (10 \leq t < 20) \\ \pi_3(t) = 1 & (t \geq 20) \end{cases}$$

4.3.3 Consumer's Contract Evaluation

In response to a proposed contract, a consumer hopes to receive a number of priced versions of the contract from agents. Each contract may offer the consumer a different range of utility values over the probability-weighted completion times. The consumer's goal is to maximize their utility, so to choose between contracts, the consumer should compute the expected utility of each contract and choose the one with greatest expected utility. All contracts based on 1 utility request should share the same target completion times.

Definition 4.3.6 (Expected utility of a contract)**.** *The expected utility of a contract* $\mathscr{C} = (\mathbf{Q}, s_D, \mathscr{T}, P, \hat{T}, \Pi)$ *with respect to utility function* $U(t, \pi)$ *is*

$$\sum_{i=1}^{n} p_i u_i(\hat{t}_i, \pi_i(\hat{t}_i))$$

when $u_i(t, \pi)$ *and* $\pi_i(t)$ *are linear functions.*

Example 4.3.7. *Suppose the consumer uses the utility function in Example 4.3.3, and two agents return two contracts* $\mathscr{C}_1$ *and* $\mathscr{C}_2$*. Further assume both agents return the same price function* Π *in Example 4.3.5, and the expected time* $\hat{T}$ *are also the same. Only the probabilities P differ as illustrated in Figure 4.3b. The consumer computes the expected utility according to Definition 4.3.6 and chooses* $\mathscr{C}_2$ *as it has greater utility.*

4.4 The Agent's Point-of-View

We now explain the agent's interactions in the market. The agent's main challenge is to assign prices to a contract, coping with the uncertainty of completion time, while taking into account the consumer's utility and the market demand. We formalize two variants of pricing (risk-aware and risk-agnostic) and formulate both as optimization problems.

4.4.1 Pricing Preliminaries

Upon receipt of the terms of a contract and the utility function of a consumer, the agent must complete the contract by computing prices for each interval and assigning probabilities to each interval.

For each configuration, we assume the financial cost C borne by the agent is a function of t: $C(t) = \alpha_C \cdot t$, where α_C is the unit rate of the configuration, and can be different across configurations. Thus, the pricing of a contract depends critically on the estimate of the completion time for $\mathbf{Q}$. Since estimates of completion time are uncertain, we model completion time T as a probability distribution over $[0, \infty)$ with probability density function

$f_T(t)$. The true $f_T(t)$ is unlikely to be known and, in practice, must be estimated by the agent with respect to a selected configuration. Based on $f_T(t)$ and $C(t)$, the agent proposes a price function $\pi(t)$, which means the consumer should pay $\pi(t)$ when the completion time is t.

The agent has three goals when pricing a contract: (i) to maintain profitability, (ii) to offer the consumer appealing utility, and (iii) to compete with the offerings of other agents. We discuss each of these goals below.

(i) **Profitability.** Naturally the agents would like to price the contract higher than their cost of execution so that they can earn a profit. Profit is uncertain for an agent because it is difficult to predict completion time in the cloud. We say a contract is *profitable in expectation* if its expected profit, with respect to the distribution $f_T(t)$, is greater than zero.

$$E[profit] = \sum_{i=1}^{n} p_i \left(\pi_i(\hat{t}_i) - C(\hat{t}_i)\right) \tag{4.1}$$

We call a contract profitable (for the agent) as long as it is profitable in expectation. The agents should always price contracts so that they are profitable, but it is possible that a particular contract ends up being unprofitable.

Definition 4.4.1 (Profitable contract)**.** *A profitable contract is a contract with* $E[profit] > 0$.

(ii) **Prioritizing consumer utility.** Since the agents knows the consumer's utility function $U(t,\pi)$ they can (and should) take it into account when choosing a configuration and pricing. To the extent that the agents can match the consumer's utility, their pricing of the contract will be more appealing to the consumer. The agents can evaluate the expected utility $E[U]$ over the distribution of time T based on their estimates and price function $\pi(t)$:

$$E[U] = \sum_{i=1}^{n} p_i u_i(\hat{t}_i, \pi_i(\hat{t}_i)) \tag{4.2}$$

Profitability for the agent and utility for the consumer are conflicting objectives: a contract that offers greater profit to the agent will offer lower utility to the consumer. We will see that the agent will attempt to maximize the consumer's utility, subject to constraints on their profitability.

(iii) **Market competitiveness and demand.** In all markets, including ours, market forces and competition prevent agents from raising prices without bound. In economics, a market demand function describes how these forces impact the pricing of goods [155].

When the agents decrease the price of a contract, the expected profit of the contract is reduced but they increase the utility of the contract to consumers. In a marketplace, when utility for the consumer increases, a greater number of consumers will accept the contract. Thus, the agents must balance the profit made from an individual contract with the overall profit they make from selling more contracts. To model this, we must make an assumption about the relationship between utility and the number of contracts that will be accepted by consumers in the market. This relationship is represented by the *demand function* which is defined as a function of utility. A linear demand curve is common in practice [155], so we focus on demand functions of the form $M(U) = a + bU$. Our framework can support demand functions of different forms, but we do not discuss these in detail.

In a real market, agents would learn about demand through repeated interactions with consumers. An agent's demand function could depend on, for example, customer loyalty, the best contracts competitors can offer, and other factors. These factors are beyond our scope. In order to simulate the functioning of a realistic market, we must assume a demand function and, for simplicity, we assume the demand functions of all agents are the same in the rest of this chapter.

4.4.2 Contract Pricing

We start from the simplest case in which the consumer has a task $\mathbf{Q}$ and a single configuration ϕ. So the cost function $C(t)$ and the pdf of the distribution of completion time

$f_T(t)$ are fixed. The agent needs to define the price function $\pi(t)$ to present a competitive contract to the consumer. Let the overall profit be $\mathscr{P}$, which equals the unit profit *profit* multiplied by the sales $M(U)$. Notice that *profit* is the profit of a single contract while $\mathscr{P}$ is the overall profit of all contracts that the agent returns to all consumers in the market. The agent wants to find the price function that leads to the greatest total profit while satisfying the profitability constraint. This results in the following optimization problem:

Problem 4.4.2 (Contract pricing)**.** *Given a contract* $\mathscr{C} = (\mathbf{Q}, s_D, \mathscr{T}, P, \hat{T}, \Pi)$*, utility function U, and demand function M, the optimal price for* $\mathscr{C}$ *is:*

$$\begin{aligned} \textit{maximize}: &\quad \mathscr{P} = E[\textit{profit}] \cdot E[M(U)] \\ \textit{subject to}: &\quad E[\textit{profit}] > 0 \end{aligned}$$

Let I_i be the interval (t_i, t_{i+1}), and recall that p_i is the probability that the completion time falls in I_i:

$$p_i = \int_{t=t_i}^{t_{i+1}} f_T(t)dt \tag{4.3}$$

Let T_i be a random variable of completion time in interval I_i. It is a truncated distribution with probability density function $f_T(t|t \in I_i)$. Let C_i be a random variable of cost in interval I_i. $C_i = C(T_i)$. So expectation $\hat{t}_i$ and expectation c_i is:

$$\hat{t}_i = E[T_i] = \int_{t \in I_i} t f_T(t|t \in I_i)dt \tag{4.4}$$

$$c_i = E[C_i] = \int_{t \in I_i} C(t) f_T(t|t \in I_i)dt \tag{4.5}$$

Therefore the expected unit profit and expected demand are:

$$E[\textit{profit}] = \sum_{i=1}^{|I|} (\pi_i - c_i) p_i \tag{4.6}$$

$$E[M(U)] = \sum_{i=1}^{|I|} M\left(U(\hat{t}_i, \pi_i)\right) p_i \tag{4.7}$$

4.4.2.1 Linear Case

When U and M are linear functions, this problem becomes a convex quadratic programming problem. It has an analytical solution. We provide an analytical solution to the linear case, in which the utility function U and demand function M are linear:

- The consumers specifies a linear utility function $U(t,\pi) = -\alpha_U \cdot t - \beta_U \cdot \pi$, which means they are always willing to pay α_U units of cost to save β_U units of time.
- The demand function is linear: $M(U) = \gamma_M + \lambda_M \cdot U$, which means that when U increased by $1/\lambda_M$, 1 more contract would be accepted. Since $U(t,\pi)$ is linear, the demand function can be written as $M(U) = \gamma_M - \alpha_M t - \beta_M \pi$.

Applying Equations 4.6 and 4.7 to Problem 4.4.2, we compute the overall profit as:

$$\begin{aligned}\mathscr{P} =& \sum_{i=1}^{|I|}(\pi_i - c_i)p_i \cdot \sum_{i=1}^{|I|} M\left(U(\hat{t}_i,\pi_i)\right)p_i \\ =& (\pi - c)^T p \cdot \left(\gamma_M - \alpha_M \hat{t}^T p - \beta_M \pi^T p\right) \\ =& -\beta_M \left(\pi^T p - \frac{\gamma_M - \alpha_M \hat{t}^T p + \beta_M c^T p}{2\beta_M}\right)^2 \\ &+ \frac{(\gamma_M - \alpha_M \hat{t}^T p - \beta_M c^T p)^2}{4\beta_M}\end{aligned}$$

$\mathscr{P}$ is maximized when

$$\pi^T p = \begin{cases} \dfrac{\gamma_M - \alpha_M \hat{t}^T p + \beta_M c^T p}{2\beta_M}, \gamma_M - \alpha_M \hat{t}^T p - \beta_M c^T p \geq 0 \\ c^T p + \varepsilon, otherwise \end{cases}$$

where ε is a small positive value.

Furthermore, when $\gamma_M - \alpha_M \hat{t}^T p - \beta_M c^T p \geq 0$,

$$\mathscr{P} = \frac{(\gamma_M - \alpha_M \hat{t}^T p - \beta_M c^T p)^2}{4\beta_M} \tag{4.8}$$

Example 4.4.3. *Let $I = [0,\infty)$; then $p_1 = 1$, and $\hat{t}_1$ is the expected completion time. The only variable is π_1. The problem becomes:*

$$\begin{aligned}
&\textit{maximize}: \\
&\qquad \mathscr{P} = (\pi_1 - c_1) \cdot (\gamma_M - \alpha_M \hat{t}_1 - \beta_M \pi_1) \\
&\textit{subject to}: \\
&\quad E[profit] = (\pi_1 - c_1) > 0
\end{aligned}$$

The result for the price π_1 is:

$$\pi_1 = max\left(\frac{\gamma_M - \alpha_M \hat{t}_1 + \beta_M c_1}{2\beta_M}, c_1 + \varepsilon\right)$$

We denote a configuration by its probability, time, and cost tuple: $(p, \hat{t}, c)$. We can apply utility function directly to this configuration as $E[U(p,\hat{t},c)] = -\alpha_U \hat{t}^T p - \beta_U \pi^T p$. We define that a configuration $(p_1, \hat{t}_1, c_1)$ is better than another configuration $(p_2, \hat{t}_2, c_2)$ when $E[U(p_1,\hat{t}_1,c_1)] > E[U(p_2,\hat{t}_2,c_2)]$. If an agent finds a better configuration than another configuration, she/he can provide consumers greater utility (defined in Equation 4.2) while making more profit (defined in Equation 4.8). Formally:

Theorem 4.4.4. *When $E[U(p_1,\hat{t}_1,c_1)] > E[U(p_2,\hat{t}_2,c_2)]$, consumer's utility $E[U_1] > E[U_2]$ and overall profit $\mathscr{P}_1 > \mathscr{P}_2$.*

Proof. We ignore the corner case in which $\gamma_M - \alpha_M \hat{t}^T p - \beta_M c^T p < 0$, which means even the most efficient configuration found by the agent leads to zero demand. When $\gamma_M - \alpha_M \hat{t}^T p - \beta_M c^T p \geq 0$, according to Section 4.4.2.1, the overall profit $\mathscr{P}$ is maximized when $\pi^T p = \frac{\gamma_M - \alpha_M \hat{t}^T p + \beta_M c^T p}{2\beta_M}$. So

$$
\begin{aligned}
E[U] &= -\alpha_U \hat{t}^T p - \beta_U \pi^T p \\
&= -\alpha_U \hat{t}^T p - \beta_U \frac{\gamma_M - \alpha_M \hat{t}^T p + \beta_M c^T p}{2\beta_M} \\
&= -\alpha_U \hat{t}^T p - \beta_U \frac{\gamma_M - \lambda_M \alpha_U \hat{t}^T p + \lambda_M \beta_U c^T p}{2\lambda_M \beta_U} \\
&= -\frac{\gamma_M}{2\lambda_M} - \frac{\alpha_U \hat{t}^T p}{2} - \frac{\beta_U c^T p}{2} \\
&= -\frac{\gamma_M}{2\lambda_M} + \frac{1}{2} E[U(p, \hat{t}, c)]
\end{aligned}
$$

So $E[U_1] > E[U_2]$ if $E[U(p_1, \hat{t}_1, c_1)] > E[U(p_2, \hat{t}_2, c_2)]$.

In addition,

$$
\begin{aligned}
\mathscr{P} &= \frac{(\gamma_M - \alpha_M \hat{t}^T p - \beta_M c^T p)^2}{4\beta_M} \\
&= \frac{(\gamma_M - \lambda_M \alpha_U \hat{t}^T p - \lambda_M \beta_U c^T p)^2}{4\beta_M} \\
&= \frac{(\gamma_M + \lambda_M E[U(p, \hat{t}, c)])^2}{4\beta_M}
\end{aligned}
$$

Since $\gamma_M - \alpha_M \hat{t}^T p - \beta_M c^T p = \gamma_M + \lambda_M E[U(p, \hat{t}, c)] \geq 0$, $\mathscr{P}_1 > \mathscr{P}_2$ if $E[U(p_1, \hat{t}_1, c_1)] > E[U(p_2, \hat{t}_2, c_2)]$. □

4.4.2.2 Selecting a Configuration

An agent typically has many available configurations for evaluating **Q**. We denote the set of configurations by $\Phi = \{\phi_1, \phi_2, ...\}$. Every configuration ϕ_j has its own cost function $C_j(t) = \alpha_{Cj} \cdot t$, where α_{Cj} is the unit rate for ϕ_j.

The agent will select the configuration that results in the most profit. The distribution of time T and its corresponding p_i, $\hat{t}_i$, and c_i then become variables in Problem 4.4.2. A naïve agent can select and enumerate a small Φ to find the best possible solution. A smarter agent will use an analytic model to solve the problem [171, 75].

4.4.3 Risk-Aware Pricing

Pricing contracts involves some risk for the agents: if their estimated distributions of time and cost are different from the actual ones, they can lose profit or even suffer losses. Next, we formally define risk based on loss and add it as part of the objective.

Definition 4.4.5 (Loss)**.** *Let the actual distribution of completion time be T^* and the optimal price function be π^*. When the agent generates a contract with price function π, the loss of revenue L is: $L_{T^*}(\pi) = \mathscr{P}(\pi^*, T^*) - \mathscr{P}(\pi, T^*) = \mathscr{P}(\pi^*, p^*, t^*, c^*) - \mathscr{P}(\pi, p^*, t^*, c^*)$, where p^* is the actual probabilities, t^* is the actual expected completion times, and c^* is the actual costs.*

There is always inherent uncertainty in the prediction of the distributions of completion of time and cost, so it is generally not possible for the agents to achieve the theoretically optimal profits based on the actual distributions. However, they can plan for this risk, and assess how much such risk they are willing to assume. We proceed to define risk as the worst-case possible loss that an agent can suffer.

Definition 4.4.6 (Risk)**.** *The risk of the agent is a function of price π, and is defined as the maximum loss over possible distributions of completion time: $R(\pi) = \max_{T^*} L_{T^*}(\pi)$.*

We incorporate risk into the agent's optimization problem by adding it to the objective function:

$$\begin{aligned} &\text{maximize} : \mathscr{P}(\pi, p, t, c) - \lambda R(\pi) \\ &\text{subject to} : E[profit] > 0 \end{aligned} \tag{4.9}$$

The parameter λ in the objective is a parameter of risk that the agent is willing to assume. Larger values of λ reduce the worst-case losses (conservative agent), while smaller values of λ increase the assumed risk (aggressive agent). The agent can estimate the risk $R(\pi)$ by solving the following optimization problem, with variables π^*, p^*, t^*, and c^*:

$$\text{maximize} : L_{T^*}(\pi) = \mathscr{P}(\pi^*, p^*, t^*, c^*) - \mathscr{P}(\pi, p^*, t^*, c^*)$$

$$\text{subject to} : E[profit^*] = \sum(\pi_i^* - c_i^*)p_i^* > 0$$

$$LBound_t \leq t_i^* - \hat{t}_i \leq UBound_t$$

$$LBound_c \leq c_i^* - c_i \leq UBound_c$$

$$0 \leq p_i^* \leq 1$$

$$\sum p_i^* = 1$$

where *LBound* and *UBound* are empirical values set by the agent. For instance, an agent's analytic model reports estimated $\hat{t_1} = 1$ min. However, the agent has executed 10 contracts and the actual mean of the time is $t_1 = 1.1$ min. The agent can set $LBound_t = 0$ and $UBound_t = 0.1$.

4.5 Fine-Grained Contract Pricing

Our treatment of pricing in Section 4.4 assumes that agents select a single configuration for the execution of a consumer contract. However, computational tasks often contain well-separated, distinct subtasks (e.g., operators in a query plan or components in a workflow). These subtasks may have vastly different resource needs. For example, Juve et al. [85] profile multiple scientific workflow systems including Montage [82] and SIPHT [109] and find that their components have dramatically different I/O, memory and computational requirements.

We now extend our pricing framework to support *fine-grained* pricing, which allows agents to optimally assign separate configurations to each subtask of a computational task. It provides more candidate contracts without changing the pricing Problem 4.4.2. Fine-grained pricing has two benefits. First, by assigning a configuration for each subtask, instead of the entire task, agents can achieve improved time and cost, resulting in higher overall utility and/or higher profit. Second, considering subtasks separately gives agents the flexibility to outsource some computation to other agents. While outsourcing computa-

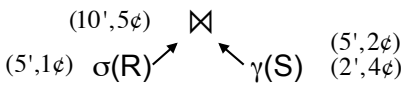


Figure 4.4: An example of a simple relational query that can be broken into 3 subtasks, corresponding to different operators.

tion across agents is not a focus of our work, it is a natural fit for our fine-grained pricing mechanism. Agents can choose to outsource subtasks to other agents based on their specialization and capabilities, or for load balancing. However, some challenges of outsourcing, such as utility and forms of contracts that agents need to exchange are beyond the scope of our current work.

We model a computational task $\mathbf{Q}$ as a directed acyclic graph (DAG) $G_{\mathbf{Q}}$. Every node in $G_{\mathbf{Q}}$ is a subtask Q_i. An edge between subtasks (Q_i, Q_j) means that the output of Q_i is an input to Q_j. When subtasks are independent of one another, the DAG may be disconnected. Our model assumes no pipelining in subtask evaluation. Therefore, a subtask Q_j cannot be evaluated until all subtasks Q_i, such that $(Q_i, Q_j) \in G_{\mathbf{Q}}$, have completed their execution.

Given the graph representation $G_{\mathbf{Q}}$ of a computational task, an agent needs to determine a configuration $\phi_i \in \Phi$ for each subtask $Q_i \in \mathbf{Q}$. This is in contrast with coarse-grained pricing (Section 4.4.2), where the agent had to select a single configuration from Φ to be used for each subtask of $\mathbf{Q}$. When the agent chooses ϕ_i, the time and cost of Q_i is $T_i(\phi_i)$ and $C_i(\phi_i)$. A set of selected configurations results in total cost $C_{\mathbf{Q}} = \sum_{Q_i} C_i(\phi_i)$, i.e., the sum of the costs of all subtasks. The completion time of $\mathbf{Q}$ is determined by the *longest* path (P) in the task graph: $T_{\mathbf{Q}} = \max_{P \in G_{\mathbf{Q}}} \sum_{Q_i \in P} T_i(\phi_i)$. Given demand M and contract utility U, $T_{\mathbf{Q}}$ and $C_{\mathbf{Q}}$ determine the agent's profit $\mathscr{P}$. The goal of the agent is to select the set of configurations that maximizes $\mathscr{P}$.

Problem 4.5.1 (Fine-grained contract pricing)**.** *Given graph $G_{\mathbf{Q}}$ representing a task $\mathbf{Q}$, and possible configurations Φ, the agent needs to specify a configuration $\phi_i \in \Phi$ for each*

$Q_i \in \mathbf{Q}$, *so that the time* $T_{\mathbf{Q}} = \max_{P \in G_{\mathbf{Q}}} \sum_{Q_i \in P} T_i(\phi_i)$ *and cost* $C_{\mathbf{Q}} = \sum_{Q_i} C_i(\phi_i)$ *maximize the overall profit* $\mathscr{P}$.

Our problem definition does not model data storage and transfer time and costs explicitly. Rather, we assume that these are incorporated in the time and cost of a subtask (T_{Q_i} and C_{Q_i}). This simplifies the model and offers an upper bound on time and cost. In practice, when two subsequent tasks share the same configuration, it is possible to reduce the costs of data passing, but these optimizations are beyond the scope of this work.

We demonstrate the intricacies of the fine-grained pricing problem through a simple example. Figure 4.4 shows a relational query with three distinct subtasks (operators): (1) select tuples from relation R, (2) aggregate on relation S, and (3) join of the results. We assume deterministic times and costs to evaluate each subtask, denoted next to each node in Figure 4.4. The select and join subtasks have only a single possible configuration each, but the aggregate subtask has two. Assume the utility function is $U(t, \pi) = -t - \pi$, which means every one minute is worth 1 cent for the consumer. Therefore, the configuration $(2', 4¢)$ is better for the aggregate subtask, since it has higher utility than the configuration $(5', 2¢)$. However, following a greedy strategy that picks the configuration that is optimal for each subtask can result in sub-optimal utility for the overall task. In this example, the join subtask has to wait 5 minutes for the select subtask to complete. Therefore, there is no benefit in paying a higher price to complete the aggregate subtask sooner, making $(5', 2¢)$ a better configuration choice.

Theorem 4.5.2. *Fine-Grained Contract Pricing is NP-hard.*

Our reduction follows from the discrete Knapsack problem.

Proof. We prove this by reducing a 0-1 Knapsack problem to it. In a knapsack problem, there are n items, each of which has a value v_i and a weight w_i. The maximum weight of a knapsack is W. One wants to maximize the sum of values V of selected items while the sum of their weights does not exceed W. We construct a contract optimization problem

Algorithm 12: Fine-Grained Contract Pricing

Require: $\mathbf{Q}, G_{\mathbf{Q}}, \Phi, \mathscr{P}(T,C)$
Ensure: *maximum* $\mathscr{P}$
1: Add node $Q_{terminal}$ with 0 time and cost to $G_{\mathbf{Q}}$
2: **for all** $Q_i \in \mathbf{Q}$ **do**
3: Add edge $(Q_i, Q_{terminal})$ to $G_{\mathbf{Q}}$
4: $\mathbf{Q}_{order} \Leftarrow TopologicalSort(G_{\mathbf{Q}})$
5: $boundT \Leftarrow$ longest time to evaluate $G_{\mathbf{Q}}$
6: **for all** $Q_i \in \mathbf{Q}_{order}$ **do**
7: $f(Q_i, 0) \Leftarrow \infty$
8: **for** $t \Leftarrow 1$ **to** $boundT$ **do**
9: $f(Q_i,t) \Leftarrow f(Q_i,t-1)$
10: **for all** $\phi \in \Phi$ **do**
11: $cost_\phi \Leftarrow \underset{q \in pred(Q_i)}{Combine}\left(f(q, t - T_i(\phi))\right) + C_i(\phi)$
12: **if** $cost_\phi < f(Q_i,t)$ **then**
13: $f(Q_i,t) \Leftarrow cost_\phi$
14: **return** $\max_t \mathscr{P}(t, f(Q_{terminal}, t))$

correspondingly. We make n subtasks in a chain. Their time and cost are deterministic. The ith subtask has two options $(w_i, v_0 - v_i)$ and $(0, v_0)$, where $v_0 = \max_i\{v_i\}$. Then let the overall profit be

$$\mathscr{P}(T,C) = \begin{cases} n * v_0 - C & T \leq W \\ 0 & T > W \end{cases}$$

So one can achieve value V in the knapsack problem without exceeding weight limit W if and only if the $\mathscr{P} = V$ in the contract optimization problem. □

We next introduce a pseudo-polynomial dynamic programming algorithm for this problem, and show that it is both efficient and effective in real-world task workflows (Section 4.7.3). Without loss of generality, we assume that time and cost are deterministic, but the algorithm can be extended to the probabilistic case in a straightforward way.

Algorithm 12 uses dynamic programming to compute the optimal profit for task graphs. The algorithm derives the exact optimal solution for cases where $G_{\mathbf{Q}}$ is a tree (e.g., relational query operators) and computes an approximation of the optimum for task graphs that are DAGs.

In Algorithm 12, $f(Q_i,t)$ represents the minimum cost for evaluating the subgraph terminated at subtask Q_i when it takes at most time t.[4] Then, $f(Q_i,t)$ can be computed based on a combination of the costs of the direct predecessors of Q_i ($pred(Q_i)$) in the task workflow (lines 7–13). When $G_{\mathbf{Q}}$ is a tree, the *Combine* function (line 11) is simply the sum of the costs of the predecessors ($\sum_{q\in pred(Q_i)} f(q,t-T_i(\phi))$), and Algorithm 12 results in the optimal profit.

If $G_{\mathbf{Q}}$ is not a tree, predecessors of a subtask Q_i can share common indirect predecessors, which introduces complex dependencies in the choice of configurations across different subtrees. For example, let $q_1,q_2 \in pred(Q_i)$, and $q_0 \in pred(q_1) \cap pred(q_2)$. Therefore, q_0 affects both subgraphs terminated at q_1 and q_2, respectively. This impacts the *Combine* function in two ways. First, the cost of q_0 should be counted only once. Second, there may be discrepancies in the configuration choice for q_0 by the different subgraphs. There are three strategies to resolve the discrepancy: (1) use the configurations with minimum time T; (2) use the configurations with minimum cost C; (3) use the configurations with maximum $\mathscr{P}(T,C)$. The *Combine* function applies the above strategies one by one, computes the time T_{Q_i} and cost C_{Q_i} of the subgraph terminated at Q_i, and updates $f(Q_i,t)$ if $T_{Q_i} \leq t$ and C_{Q_i} is better. Note that Strategy 1 guarantees a feasible solution whenever one exists.

4.6 Alternative Approaches

4.6.1 Benchmark-Based Approach

Floratou et al. [54] propose a Benchmark as a Service (BaaS) approach to help consumers select configurations. This new BaaS benchmarks user's workload and use the optimal configuration to execute the workload repetitively. As they mention in the paper, changes such as growth of input data make BaaS complicated. So a BaaS provider need to monitor and react to these changes. In our approach, we do not make assumptions about

[4] We turn the continuous space of time t into discrete space by choosing an appropriate granularity (e.g., minute).

the repetitiveness of workloads. The disadvantage is that consumers may pay more for repetitive workloads even when they are very similar. The advantage is that consumers do not need to worry about the change of the workloads.

We compare with the benchmark-based approach in Section 4.7.4.

4.6.2 VCG-Auction-Based Approach

VCG auction is a pricing mechanism. Its strategy-proof property makes it popular in many studies. We develop a VCG auction model and compare it with our approach. In this VCG model, a customer opens a bidding and agents bid on prices. Notice that this model defines consumers payment according to the utility instead of the pure price, which is different from the canonical VCG mode.

Assume agent i proposes its contract with utility $Util_i$. The consumer takes the contract with the highest utility $Util_i$ but pays based on the second highest utility $U^* = \max_{j \neq i}(U_j)$. The payment is a piecewise function $\Omega(t) = \omega_k(t)$. From agent i's perspective, its cost function is $C_i(t)$, so its payoff is:

$$payoff_i = \begin{cases} E[profit] = \sum_{k=1}^{n} p_k(\omega_k(\hat{t_k}) - C_i(\hat{t_k})) & \text{if } U_i > \max_{j \neq i}(U_j) \\ 0 & otherwise \end{cases}$$

Here is the definition of $\Omega(t)$ when $U(t,\pi) = -\alpha_U t - \beta_U \pi$ is linear: Let $\Delta = U_i - U^*$. The inverse function of $U(t,\pi)$ is $\Pi(t,u) = (-\alpha_U t - u)/\beta_U$. We define $\Omega(t) = \Pi(t, u - \Delta) = \Pi(t,u) + \Delta/\beta_U$.

Example 4.6.1. *Suppose the utility function is* $U(t,\pi) = -t - 2\pi \ \ (0 < t < \infty)$. *So* $\Pi(t,u) = (-t-u)/2$. *When* $\Delta = U_i - U^* = 10$, $\Omega(t) = \Pi(t, u-\Delta) = (-t-(u-10))/2 = (-t-u)/2 + 5$.

Theorem 4.6.2. *When* $U(t,\pi)$ *is linear, the above* $\Omega(t)$ *satisfies:*

1) $U_i > U^* > U_i^C \Rightarrow E[\text{profit}] < 0$;

2) $U_i^C > U^* > U_i \Rightarrow E[\text{profit}] > 0$.

Given the payment function, we can show that our developed VCG auction is strategyproof.

Proof.

1) $\Delta = U_i - U^*$ and $U_i > U^* > U_i^C$

$\Rightarrow \Delta < U_i - U_i^C = \sum_{k=1}^{n} p_k[U(\hat{t_k}, \pi_k) - U(\hat{t_k}, c_k)]$

$\Rightarrow \sum_{k=1}^{n} p_k[U(\hat{t_k}, \pi_k) - U(\hat{t_k}, c_k)] - \Delta > 0.$

$0 \leq p_k \leq 1$ and $\sum_{k=1}^{n} p_k = 1$

$\Rightarrow \Delta = \sum_{k=1}^{n} p_k \Delta.$

So $\sum_{k=1}^{n} p_k[U(\hat{t_k}, \pi_k) - \Delta - U(\hat{t_k}, c_k)] > 0$

$\Rightarrow \sum_{k=1}^{n} p_k(-\beta_U \pi_k - \Delta + \beta_U c_k) > 0.$

So $E[profit] = \sum_{k=1}^{n} p_k(\omega_k(\hat{t_k}) - C_i(\hat{t_k})) = -\sum_{k=1}^{n} p_k(-\beta_U \pi_k - \Delta + \beta_U c_k) < 0.$

Therefore $U_i > U^* > U_i^C \Rightarrow E[profit] < 0$;

2) Similar to above, we can prove $U_i^C > U^* > U_i \Rightarrow E[profit] > 0$. □

Theorem 4.6.3. *Every agent truthfully revealing its cost is a weakly-dominant strategy.*

Proof. Truthfully bidding means $U_i = U_i^C$.

1) The strategy of overbidding, $U_i > U_i^C$, is dominated by truthfully bidding.

When $U^* > U_i > U_i^C$, both strategies yield $payoff_i = 0$.

When $U_i > U_i^C > U^*$, both strategies yield $payoff_i = E[profit] > 0$.

However, when $U_i > U^* > U_i^C$, overbidding yields $payoff_i = E[profit] < 0$ (Theorem 4.6.2) while truthfully bidding yields $payoff_i = 0$.

So overbidding is dominated by truthfully bidding.

2) The strategy of underbidding, $U_i < U_i^C$, is dominated by truthfully bidding.

When $U^* > U_i^C > U_i$, both strategies yield $payoff_i = 0$.

When $U_i^C > U_i > U^*$, both strategies yield $payoff_i = E[profit] > 0$.

However, when $U_i^C > U^* > U_i$, underbidding yields $payoff_i = 0$ while truthfully bidding yields $payoff_i = E[profit] > 0$ (Theorem 4.6.2).

So underbidding is dominated by truthfully bidding.

So truthfully bidding is a weakly-dominant strategy. □

Theorem 4.6.4. *Every agent truthfully revealing its cost is a weakly-dominant strategy.*

In other words, every rational agent will make its cost be the price in its contract. This proof is very similar to the proof for the canonical VCG auction.

Our posted-price model and VCG auction model both exist in the real world market. We discussed in the related work section that neither of them dominates the other. They have 2 main differences in our case: 1) Posted-price model requires the agent to better understand the demand function of the market. Then an agent can set the price actively to gain more profit. In contrast, agents in VCG auction only needs to truthfully reveal their costs, then the price is passively decided based on the second best utility. 2) VCG auction requires a centralized auctioneer who ensures the consumers pay according to the second best utility. It makes a cross-platform market more difficult to form. Posted-price model does not have such requirement.

We quantitatively compare with the VCG-auction-based approach in Section 4.7.4.

4.6.3 Differentiated Bertrand

Multiple agents competing in the market is a typical differentiated Bertrand model. Specifically, Bertrand model solves the equilibrium of optimal prices based on all agents' demand functions. But in practice, an individual agent can hardly know other agents' demand function when pricing in the market. So our model assumes that each agent observes a demand function of its own price. Such demand function is valid given the other agents' current prices. An agent will change its price when others change. The prices will converge to the equilibrium in the long term. Now we show the connection through an example.

In a differentiated Bertrand model, suppose there are 2 agents numbered 1 and 2. Their prices are μ_1 and μ_2. Agent 1's demand function is $M_1(\mu_1,\mu_2) = \gamma - \alpha\mu_1 + \beta\mu_2$ where α,β,γ are positive parameters. Higher μ_1 means lower M_1 but higher μ_2 means higher M_1 due to competition. Similarly, $M_2(\mu_1,\mu_2) = \gamma - \alpha\mu_2 + \beta\mu_1$. So the overall profit $\mathscr{P}_1 = M_1(\mu_1,\mu_2)\cdot\mu_1$ [5], $\mathscr{P}_2 = M_2(\mu_1,\mu_2)\cdot\mu_2$. Given a fixed μ_1, the best μ_2 that maximizes $\mathscr{P}_2$ is:

$$\mu_2^* = (\gamma + \beta\mu_1)/2\alpha. \tag{4.10}$$

Similarly, the best μ_1 is:

$$\mu_1^* = (\gamma + \beta\mu_2)/2\alpha. \tag{4.11}$$

So one can solve these two equations to get a Nash Equilibrium: $\mu_1^* = \mu_2^* = \gamma/(2\alpha - \beta)$.

In the real world, agents' demand functions cannot be exactly the same. Thus we should use different demand functions $M_1(\mu_1,\mu_2) = \gamma_1 - \alpha_1\mu_1 + \beta_1\mu_2$ and $M_2(\mu_1,\mu_2) = \gamma_2 - \alpha_2\mu_1 + \beta_2\mu_2$. So $\mu_1^* = (2\alpha_2\gamma_1 + \beta_1\gamma_2)/(4\alpha_1\alpha_2 - \beta_1\beta_2)$ and $\mu_2^* = (2\alpha_1\gamma_2 + \beta_2\gamma_1)/(4\alpha_1\alpha_2 - \beta_1\beta_2)$.

The above calculation, however, requires that both agents know both demand functions M_1 and M_2, which may not be a realistic assumption; one may easily obtain one's own demand function by fitting historical data, but it may not be possible to know the other party's demand function. Without knowledge of the other party's demand function, one generally cannot settle for the NE (μ_1^*,μ_2^*) in one shot, but has to adjust one's price dynamically according to the observed demand function. Thus we have a repeated game here, and rational agents will follow the best response functions 4.10 and 4.11. Our approach uses exactly the same response functions, where the impact of the other agent's price is absorbed into the intercept. More precisely, in our model, Agent 1 tries to optimize $M_1(\mu_1) = \gamma_1' - \alpha_1\mu_1$ where $\gamma_1' = \gamma_1 + \beta_1\mu_2$, so it sets $\mu_1 = \gamma_1'/2\alpha_1 = (\gamma_1 + \beta_1\mu_2)/2\alpha_1$. Similarly, Agent 2 sets

[5] The Bertrand model in [136] assumes marginal cost $c = 0$ for simplicity. So the price in [136] corresponds to the profit in our approach. i.e. $\mu + c = \pi$.

$\mu_2 = (\gamma_2 + \beta_2\mu_1)/2\alpha_2$. If both agents keep updating their prices in this manner, their prices will eventually converge to the NE (μ_1^*, μ_2^*).

Recall that in the Differentiated Bertrand Model with nonidentical demand functions, the best response functions are given by

$$\mu_1 = f_1(\mu_2) = a_1\mu_2 + b_1, \tag{4.12}$$

$$\mu_2 = f_2(\mu_1) = a_2\mu_1 + b_2, \tag{4.13}$$

where

$$a_i = \frac{\beta_i}{2\alpha_i}, \quad b_i = \frac{\gamma_i}{2\alpha_i}, \quad \text{for } i = 1,2.$$

If $a_1a_2 < 1$, there exists a Nash Equilibrium (μ_1^*, μ_2^*), which is the unique solution to the following equations,

$$\begin{cases} \mu_1^* = f_1(\mu_2^*), \\ \mu_2^* = f_2(\mu_1^*). \end{cases}$$

Suppose that both agents keep updating their prices to the best response to the currently observed price of the other party. We show that their prices eventually converges to the NE (μ_1^*, μ_2^*).

We allow for asynchronous updates. Thus at the x-th update, there are three possibilities,

(1) Only agent 1 updates his prices, in which case

$$(\mu_{1,x+1}, \mu_{2,x+1}) = F_1(\mu_{1,x}, \mu_{2,x}) = (f_1(\mu_{2,x}), \mu_{2,x}). \tag{4.14}$$

(2) Only agent 2 updates his prices, in which case

$$(\mu_{1,x+1}, \mu_{2,x+1}) = F_2(\mu_{1,x}, \mu_{2,x}) = (\mu_{1,x}, f_2(\mu_{1,x})). \tag{4.15}$$

(3) Both agents update their prices, in which case

$$(\mu_{1,x+1}, \mu_{2,x+1}) = F_3(\mu_{1,x}, \mu_{2,x}) = (f_1(\mu_{2,x}), f_2(\mu_{1,x})). \tag{4.16}$$

Thus $(\mu_{1,x}, \mu_{2,x}) = G_x \circ G_{x-1} \circ \cdots \circ G_1(\mu_{1,0}, \mu_{2,0})$, where $\mu_{1,0}$ and $\mu_{2,0}$ are the initial prices, and $G_i \in \{F_1, F_2, F_3\}$ for $i = 1, 2, \ldots, x$. We assume that both agents keep updating their prices, i.e.,

$$\lim_{x\to\infty} \sum_{i=1}^{x} \mathbf{1}\{G_i \neq F_1\} = \lim_{x\to\infty} \sum_{i=1}^{x} \mathbf{1}\{G_i \neq F_2\} = \infty. \tag{4.17}$$

Theorem 4.6.5. *Assume $a_1 a_2 < 1$. If both agents keep updating their prices, i.e.* (4.17) *holds, then for any initial prices $\mu_{1,0}$ and $\mu_{2,0}$, we have*

$$\lim_{x\to\infty} (\mu_{1,x}, \mu_{2,x}) = (\mu_1^*, \mu_2^*).$$

Proof. For any function f, let $f^{(0)} = \mathrm{id}$, the identity function, and $f^{(m)} = f^{(m-1)} \circ f$ for $m \geq 1$. Note that $F_i^{(2)} = F_i$ and $F_3 \circ F_i = F_{3-i} \circ F_i$ for $i = 1, 2$. By repeated applications of these relations, we obtain

$$(\mu_{1,x}, \mu_{2,x}) = F_2^{(m_2)} \circ H_1^{(m)} \circ F_1^{(m_1)} \circ F_3^{(m_3)}(\mu_{1,0}, \mu_{2,0}), \tag{4.18}$$

where $H_1 = F_1 \circ F_2$, $m_1 \in \{0, 1\}$, $m_2 \in \{0, 1\}$ and $m_1 + m_2 + m_3 + m = N_x$. Note that N_x is the number of *effective* updates that can potentially change the prices. Without loss of generality, we assume that $N_x = x$, which amounts to discarding those updates that cannot change the price of either agent.

- Case I: $m_1 = m_2 = m = 0$

In this case, both agents always synchronize their updates and

$$(\mu_{1,x},\mu_{2,x}) = F_3^{(x)}(\mu_{1,0},\mu_{2,0})$$
$$= \begin{cases} (g_1^{(\frac{x}{2})}(\mu_{1,0}), g_2^{(\frac{x}{2})}(\mu_{2,0})), & \text{for } x \text{ even}, \\ (g_1^{(\frac{x-1}{2})} \circ f_1(\mu_{2,0}), g_2^{(\frac{x-1}{2})} \circ f_2(\mu_{1,0})), & \text{for } x \text{ odd}, \end{cases}$$

where $g_1 = f_1 \circ f_2$ and $g_2 = f_2 \circ f_1$. Note that $g_1(\mu_1^*) = \mu_1^*$, and, for any z,

$$g_1(z) - \mu_1^* = g_1(z) - g_1(\mu_1^*) = a_1[f_2(z) - f_2(\mu_1^*)] = a_1 a_2 (z - \mu_1^*).$$

Thus for any z,

$$|g_1^{(k)}(z) - \mu_1^*| = (a_1 a_2)^k |z - \mu_1^*| \to 0, \quad \text{as } k \to \infty.$$

Similarly, for any z,

$$|g_2^{(k)}(z) - \mu_2^*| = (a_1 a_2)^k |z - \mu_2^*| \to 0, \quad \text{as } k \to \infty. \tag{4.19}$$

It follows that

$$\lim_{x\to\infty} (\mu_{1,x},\mu_{2,x}) = (\mu_1^*,\mu_2^*).$$

- Case II: $m_1 + m_2 + m > 0$

  In this case, the agents do not always synchronize their updates and $m \to \infty$ in (4.18). By symmetry, we assume $m_1 = 1$; the other case can be dealt with similarly. Note that

  $$H_1^{(m)} \circ F_1(\mu_1,\mu_2) = (f_1 \circ g_2^{(\lfloor \frac{m}{2} \rfloor)}(\mu_2), g_2^{(\lceil \frac{m}{2} \rceil)}(\mu_2)).$$

  Let $(\mu_1,\mu_2) = F_3^{(m_3)}(\mu_{1,0},\mu_{2,0})$. By (4.19),

  $$\lim_{m\to\infty} H_1^{(m)} \circ F_1(\mu_1,\mu_2) = (f_1(\mu_2^*),\mu_2^*) = (\mu_1^*,\mu_2^*).$$

Type	CPU (virtual)	Memory	$/hour
db.m3.Medium	1	3.75GB	$0.095
db.m3.Large	2	7.5GB	$0.195
db.m3.xLarge	4	15GB	$0.390
db.m3.2xLarge	8	30GB	$0.775
db.r3.Large	2	15GB	$0.250
db.r3.xLarge	4	30.5GB	$0.500
db.r3.2xLarge	8	61GB	$0.995
m1.Medium	1	3.75GB	$0.109
m1.Large	2	7.5GB	$0.219
m1.xLarge	4	15GB	$0.438

Figure 4.5: Types of Amazon machines and associated features and costs (in January 2015). The first 7 types (db.*) are RDS configurations, whereas the last 3 (m1.*) are EMR configurations. The prefixes (db and m1) are omitted from some figures for brevity.

It follows that

$$\lim_{x\to\infty}(\mu_{1,n},\mu_{2,n}) = \lim_{m\to\infty} F_2^{(m_2)} \circ H_1^{(m)} \circ F_1(\mu_1,\mu_2)$$
$$= F_2^{(m_2)}(\mu_1^*,\mu_2^*) = (\mu_1^*,\mu_2^*),$$

where in the last step we have used the fact that (μ_1^*,μ_2^*) is a fixed point of $F_2^{(m_2)}$ for $m_2 = 0,1$. □

4.7 Experimental Evaluation

In this section we evaluate our market using a real-world cloud computing platform: Amazon Web Services (AWS). Our experiments collect real-world data from a variety of relational and MapReduce task workloads, and use this data to simulate the behavior of our market entities on the AWS cloud. Our results demonstrate that our market framework offers incentives to consumers, who can execute their tasks more cost-effectively, and to agents, who make profit from providing fair and competitive contracts.

We proceed to describe our experimental setup, including computational tasks, consumer parameters, and contracts.

1. **Data and configurations.** We spent 8,106 machine hours and $3,118 in obtaining the distributions of time and cost for two types of computational tasks: relational query workloads, and MapReduce jobs.

- **Relational query tasks:** We use the queries and data of the TPC-H benchmark to evaluate relational query workloads. We use all 22 queries of the benchmark on a 5GB dataset (scale factor 5). We use the Amazon Relational Database Service (RDS) to evaluate the workloads on 7 machine configurations, each of which has 200GB of Provisioned IOPS SSD storage, and runs PostgreSQL 9.3.5. Figure 4.5 lists the capacity and hourly rate of each configuration.
- **MapReduce tasks:** We evaluate MapReduce workloads using three job types (WordCount, Sort, and Join) over 5GB of randomly generated input data. We use the Amazon Elastic MapReduce service (EMR) to test our framework on these workloads. We select 3 machine configuration types. Figure 4.5 lists the capacities and hourly rates of these configurations. We experimented with 4 different sizes of clusters for each machine configuration: 1, 5, 10, and 20 slave nodes.
- **Scientific workflows:** We use real-world scientific workflows that represent computational tasks with multiple subtasks, to evaluate fine-grained pricing (Section 4.5). We retrieved 1,454 workflows from MyExperiment [40], one of the most popular scientific workflow repositories. These workflows were developed using Taverna 2 [168], and comprise the majority of workflows in the repository. The size of workflows ranges from 1 to 154 subtasks.

2. **Consumer models.** We simulate the consumer behavior in our framework using the utility and demand functions.

- **Utility:** In our evaluation, utility is a linear function $U(t,\pi) = -\alpha_U t - \beta_U \pi$ modeling consumer preferences. α_U represents the unit cost that the consumer is willing to pay to save β_U unit time. For our experiments, we assume $U(t,\pi) = -t - \pi$, where t is

measured in minutes and π is measured in cents, which means every minute is worth 1 cent to the consumer.

- **Demand:** In our evaluation, the demand function is linear: $M(U) = \gamma_M + \lambda_M U$, which means that when U increases by $1/\lambda_M$, 1 more contract would be accepted. We use $M(U) = 100 + 50U$ for RDS, and $M(U) = 100 + 5U$ for EMR. λ_M is smaller for EMR because the times and costs for MapReduce jobs are much larger than those of relational queries.

3. **Contracts.** All our experiments involve contracts with a deadline, which means that every consumer request specifies one target completion time. We execute each task 100 times using every configuration and set the deadline of each query as the average completion time across all configurations.

4.7.1 Consumer Incentives

In this section, we evaluate whether our market framework offers sufficient incentive for consumers to participate in the market. Our first set of experiments simulates several naïve cloud users who select one of the default configurations for their computational tasks: 7 configurations for RDS, and 12 configurations for EMR. Then we simulate a baseline user who intuitively chooses configurations based on a simple feature of a task. Specifically, the user chooses a configuration with the best CPU performance for a CPU-intensive task, or a configuration with the best IO performance for an IO-intensive task. Predicting whether a task is CPU-intensive or IO-intensive is a difficult task for a user. However, we unfairly bias toward this baseline by indeed executing the task to measure its CPU time and IO time. We consider a task is CPU-intensive if its CPU time is greater, otherwise IO-intensive. Finally we simulate an expert agent, who, for every task, selects the configuration that maximizes the consumer's utility function. Figure 4.6a presents the price and time achieved by each of the 7 default configurations for RDS, as well as the price and time offered by the expert agent. The line in the graph is the utility indifference curve for the agent's configuration,

representing points with the same utility value. Points on the curve are equally good, from the consumer's perspective, as the one achieved by the expert agent. Points above the curve have worse utility values (less preferable than the agent's offer), while points below the curve have better utility values (more preferable than the agent's offer).

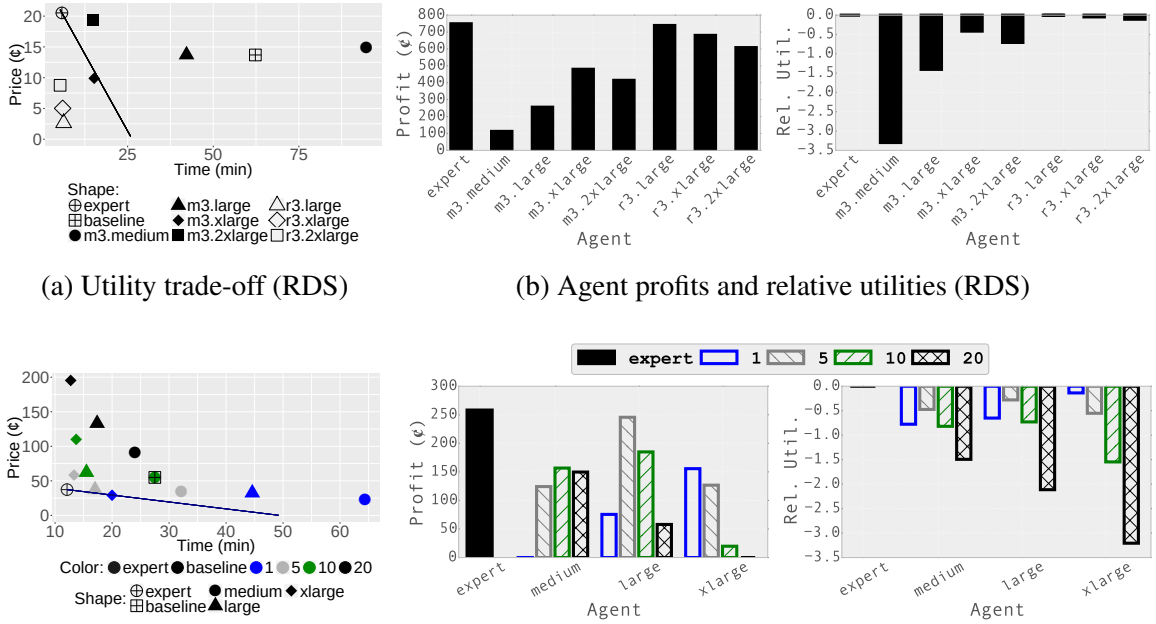
Our experiments show that the expert agent provides more utility to 4 out of 8 naïve cloud users with relational query tasks on RDS. This means that, even though the agent makes a profit, a good portion of the users would still benefit from using the market instead of relying on default settings. This effect is even more pronounced for EMR workloads. Figure 4.6c shows that the expert agent offers better utility to *all* simulated naïve users. This means that, in every single case, the consumers get better utility by using the agent's services, instead of selecting a default configuration. It is noteworthy that the heuristic-based baseline approach is 189% and 67% worse in utility than our approach for RDS and EMR workloads, respectively.

4.7.2 Agent Incentives and Market Properties

In this section we demonstrate that the pricing framework satisfies three important properties: *competitiveness*, *fairness*, and *resilience*. These properties incentivize consumers and agents to use and trust the market by ensuring that (a) the agents will identify efficient computation plans and provide accurate pricing, and (b) inaccurate estimates will not pose a great risk to the agents.

4.7.2.1 Competitiveness

We run experiments on Amazon RDS and EMR to demonstrate how different configurations impact profitability in practice. Our goal is to show that, in our market, well-informed, expert agents can make more profit than naïve agents, thus creating incentives for agents to be competitive and offer configurations and contracts that benefit the consumers. In this experiment, a naïve agent selects one configuration to use for all queries. In contrast, the expert agent always selects the optimal configuration for each query. The goal of this



(a) Utility trade-off (RDS)

(b) Agent profits and relative utilities (RDS)

(c) Utility trade-off (EMR)

(d) Agent profits and relative utilities (EMR)

Figure 4.6: (a, c) Users achieve better utility by using an expert agent, compared to naïvely selecting a default configuration. The agent benefits 40% of the consumers in RDS workloads, and 100% of the consumers in EMR workloads. (b, d) Expert agents always achieve the largest profits. This means that our market framework gives incentives to agents to find optimal configurations.

experiment is to show the impact of configuration selection. Thus we control for other parameters, such as the accuracy of the agents' estimates. So, for now, we assume that all agents know the distributions of time and cost accurately. We relax this assumption in later experiments.

We generate histograms of time and cost by evaluating each query with each configuration 100 times. All agents use these histograms to approximate the distributions and price contracts based on these distributions. After an agent prices a contract, we compute the number of accepted contracts according to the demand function, $M(U)$. Then we randomly select M executions to do trials. The agent receives payments based on whether the execution met or missed the deadline.

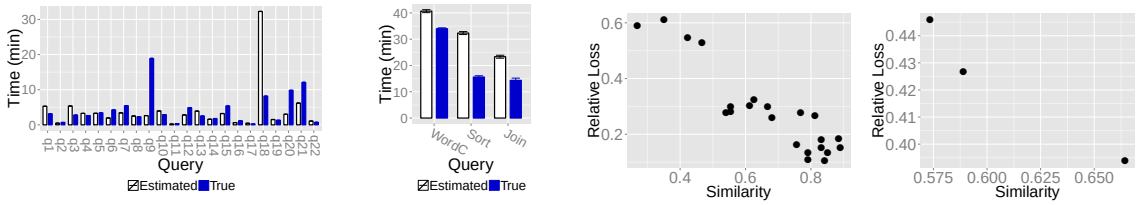
	Task	Configuration
RDS	q1–q3, q5–q16, q18, q22	db.r3.Large
	q4, q17, q19, q20	db.r3.xLarge
	q21	db.r3.2xLarge
EMR	WordCount	m1.Medium × 20
	Sort	m1.Large × 5
	Join	m1.xLarge × 1

Figure 4.7: The expert agents select different configurations for different tasks to maximize profit.

Figure 4.6b illustrates the total profit made by each agent pricing RDS workloads. There are 7 naïve agents, each using one of the RDS configurations from Figure 4.5, and one expert agent, who always uses the best configuration for each task. Figure 4.6d illustrates the same experiment on EMR workloads. We use one expert agent and 12 naïve agents who used the three EMR configurations from Figure 4.5, each with a cluster size of 1, 5, 10, or 20 nodes. Figure 4.7 lists the configuration chosen by the expert agent for each RDS and each EMR task. In both experiments, the expert agent achieves the *highest* overall profit.

Figures 4.6b and 4.6d also show the utilities offered by the agents for the same contracts. We plot the relative utility of each naïve agent, using the utility of the expert agent as a baseline: $\frac{AgentUtility - ExpertUtility}{|ExpertUtility|}$. On both RDS and EMR workloads, the utility offered by the expert was the best among all agents.

Our experiments on both RDS and EMR demonstrate that expert agents achieve better utility and profit than all other agents. This verifies empirically that our market design ensures incentives for agents to improve their estimation techniques and configuration selection mechanisms. This benefits both consumers, who get better utility, and agents, who get more profit.



(a) Estimate (RDS) (b) Estimate (EMR) (c) Fairness (RDS) (d) Fairness (EMR)

Figure 4.8: Agents' estimates are often inaccurate, and such inaccuracies can lead to loss of profit.

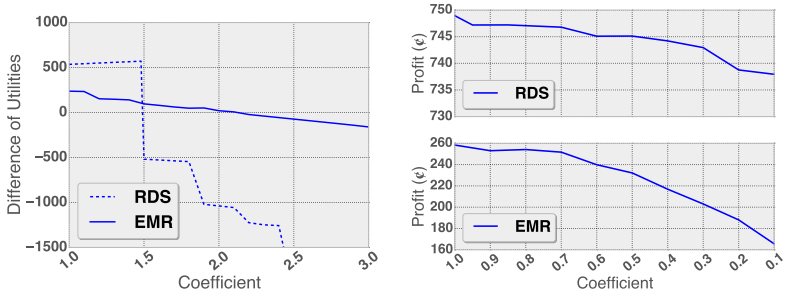
4.7.2.2 Fairness

Fairness guarantees the incentive for agents to present accurate estimates to consumers. If the agent uses inaccurate estimates, she/he will be penalized with lower profits. Our goal is to show that more accurate estimates lead to greater profit for the agent in practice.

We consider an agent using db.m3.medium on RDS and PostgreSQL's default query optimizer to estimate the completion times of queries. The PostgreSQL optimizer provides an estimate of the expected completion time and the agent assumes a Gaussian distribution with a mean value equal to the completion time predicted by the optimizer. We chose 0.05 for the standard deviation, which is very close to the actual average standard deviation of the distributions of the 22 TPC-H queries (0.04).

We also consider another agent using m1.medium on EMR, with one master and one slave node. The agent estimates the expected completion time by executing queries on a 5% sample of the data, and assumes a Gaussian distribution around the estimated mean. The agent uses an empirical standard deviation, 0.55, which is close to the average true standard deviation of all three EMR job types (0.56).

We compare the agents' estimate with the true distributions in Figures 4.8a and 4.8b. We plot the average completion time for each TPC-H query and each EMR task. The standard deviation is very low (under 0.75 min) for all tasks. As these plots show, the agents' estimates can often be far from the actual completion times (e.g., q_{18}).



(a) (Agent's utility - consumer's utility) drops in overestimation.

(b) Agent's profit decreases moderately in underestimation.

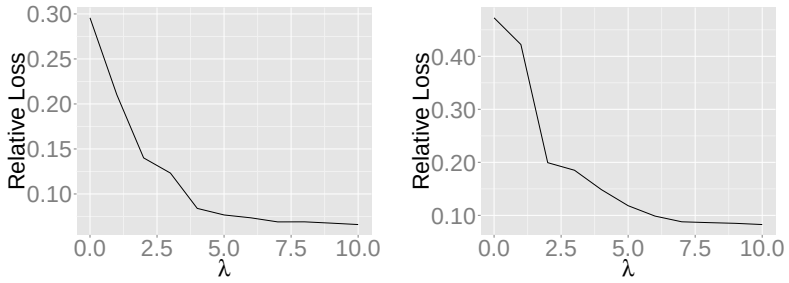
Figure 4.9: Poor estimation moderately impacts the market.

Next, we use the similarity between two distributions and relative loss to show the relationship between estimation accuracy and profit. We compare the true distribution of completion time (which is a histogram) with the agent's estimate (a Gaussian distribution) by turning the agent's estimate into a histogram and computing the cosine similarity between two histograms. The relative loss measures how much profit the agents lose compared to the optimal profit they could have made. We define relative loss of profit as:

$$RelativeLoss = \frac{OptimalProfit - ActualProfit}{OptimalProfit} \tag{4.20}$$

As Figures 4.8c and 4.8d illustrate, when the agent's estimate is more accurate, the relative loss is smaller.

Our market does not rely on the assumption that the estimates are accurate, and it can in fact tolerate inaccuracies well. As long as there exists at least one task for which an agent can produce better estimates than a consumer, the agent offers utility to the market. In our experimental evaluation, we showed that this is easy to achieve in practice: even agents using simple estimation methods (such as using the PostgreSQL optimizer or sampling), which result in fairly inaccurate estimates, can provide benefit to non-expert consumers. Existing research has shown that time and cost estimation is non-trivial [2, 75, 171], and



(a) Resilience (RDS) (b) Resilience (EMR)

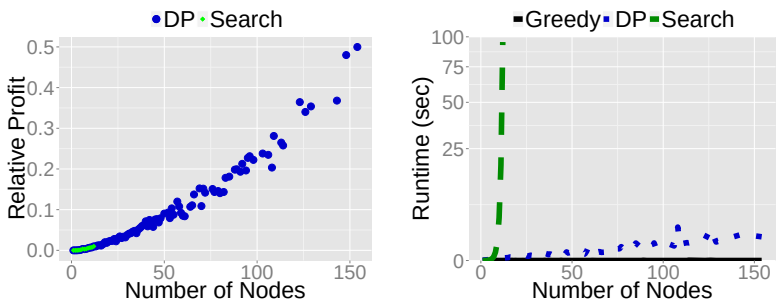
Figure 4.10: By adjusting for risk, agents can reduce their losses in case of inaccurate estimates.

agents using such specialized tools would always produce better estimates than non-expert consumers.

In addition, we further expanded our evaluation to study an extreme case: when all agents in the market make worse estimation than all consumers, for all tasks. We multiply the agents' estimated time and cost by a coefficient. A > 1 coefficient means overestimation and a < 1 coefficient means underestimation.

As depicted in Figure 4.9a, overestimation leads agents to post higher prices lowering the consumers' utilities. However, switching to using the cloud provider directly becomes preferable (on average) only when agents overestimate substantially: in our empirical simulation, agents had to overestimate time and cost by 49% in RDS workloads, and by 120% in EMR workloads before a switch was beneficial to consumers on average.

On the other hand, figure 4.9b shows that underestimation of time and cost decreases an agent's profit by 2% in RDS and 36% in EMR if it underestimates time and cost by a factor of 10. Depending on the agents' profit margins, they may be able to absorb the difference without losses. To avoid losses, agents can follow risk-aware pricing strategies (Section 4.4.3).



(a) Profit Relative to Greedy (b) Runtime

Figure 4.11: DP outperforms Greedy and Search

4.7.2.3 Resilience

The property of *resilience* provides assurances to the agents, by ensuring that inaccurate estimates will not pose a significant risk to the agents' profits. This property is crucial, as errors in the estimates are very common [6, 48, 171]. Our framework ensures resilience to these inaccuracies by accounting for *risk* (Definition 4.4.6). Specifically, the agents can profit by adjusting the risk they prefer to take. According to Equation 4.9, the risk is part of the objective and controlled by a parameter λ. When λ is large, the agent has low confidence in the estimate (conservative). This setting reduces the loss of profit if the agent's estimate is inaccurate.

We again consider an RDS agent using db.m3.medium and the default PostgreSQL optimizer, and an EMR agent using m1.medium and sampling to estimate runtime. We evaluate relative loss using Equation 4.20 and plot it for different values of λ (Figure 4.10). A value of $\lambda = 0$ means that the agent is confident that their estimate is correct. However, since in this case the estimates were inaccurate, the relative loss for $\lambda = 0$ is high: the agents' profit is much lower than the optimal profit they could have achieved. For both agents (EMR and RDS), the relative loss decreases for higher values of λ. This shows that by adjusting for risk, the agents can reduce loss of profit.

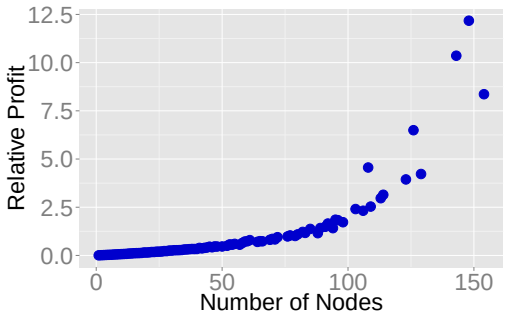


Figure 4.12: Pricing at finer granularities can vastly increase the agents' profits.

4.7.3 Fine-Grained Pricing

In our final set of experiments, we evaluate fine-grained pricing (Algorithm 12) against a large dataset of real-world scientific workflows [40]. This dataset is well-suited for this experiment, as it provides diverse computational flows of varied sizes and complexities. The published workflows do not report real execution information (time and cost), and we are not aware of any public workflow repositories that provide this information. Therefore, we augment the real workflow graphs with synthetic time and cost histograms for each subtask, drawn from random Gaussian distributions with means in the [1,100] range, and variances in the [0,5] range. Each subtask has 5 candidate configurations with different time and cost histograms. We compute the profit using utility $U(t,\pi) = -t - \pi$ and demand $M(U) = 100 + 0.01U$ (Section 4.4.2.1). We set λ_M (the coefficient of U) to a smaller value than the ones used for RDS and EMR workloads, because the completion times and costs for workflows are much larger.

First, we evaluate our Dynamic Programming algorithm (Algorithm 12) against two baselines: (1) an exhaustive search strategy (*Search*) that explores all possible configuration assignments, and (2) a greedy strategy (*Greedy*) that selects the configuration that leads to the maximum local profit for each subtask. We perform 10 repetitions for each workflow, using different random time and cost distributions for each repetition. Figure 4.11a shows the relative profit achieved by *Search* and *DP* compared to *Greedy*: $\frac{\mathscr{P} - \mathscr{P}_{Greedy}}{\mathscr{P}_{Greedy}}$. *DP* achieves better profits than *Greedy*, and the effect increases for larger workflows: for workflows

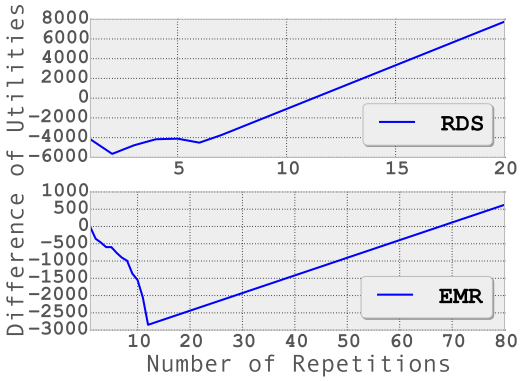


Figure 4.13: Consumers prefer our approach to benchmark except for highly repetitive workloads

with 154 subtasks, *DP* achieves 50.0% higher profit than *Greedy*. *Search* provides few data points, as it cannot scale to larger graphs. For small workflows (up to 12 subtasks) *Search* and *DP* select equivalent configurations that result in the same (optimal) profit. Figure 4.11b shows the running time of the three algorithms. As expected, exhaustive search quickly becomes infeasible, and *Greedy* is faster than *DP*. However, the runtime of *DP* remains low even for larger workflows. Combined with the profit gains over *Greedy*, this experiment demonstrates that Algorithm 12 is highly effective for fine-grained pricing.

Second, we evaluate the benefits of fine-grained pricing, compared to coarse-grained pricing. Figure 4.12 shows the profit achieved by *DP*, which assigns a configuration to each subtask, relative to the optimal single configuration for the entire workflow. In this experiment, fine-grained pricing doubled the agents' profits for small workflows, compared to coarse-grained pricing, and the gains increase as workflows grow larger. For the largest workflows in our dataset, fine-grained pricing achieved 12.5 times higher profits.

4.7.4 Comparison with Alternative Approaches

4.7.4.1 Benchmark-based Approach

Contrasting our work with Benchmarking as a Service (BaaS) [54] is meaningful when workload repetition is significant. We assume a consumer who repeate RDS and EMR

workloads without any modifications, and with each repetition tested a different configuration; once all configurations were tested, the consumer would continue using the best one in subsequent repetitions. For this experiment, we limited the number of possible configurations to 7 for RDS and 12 for EMR. This biases the experiment in favor of benchmarking, as in practice the number of configurations that the consumer would have to try is much higher. In this simplified setting, we found that it took 12 repetitions in RDS and 68 repetitions in EMR before the consumer would start benefiting from benchmarking. In the real world, these numbers are much higher, as cloud providers offer way more configurations than the ones we considered here. Cluster size alone causes an explosion in the number of options, so having an agent with an analytical model, such as in [75] , is necessary.

In practice, BaaS has additional challenges: As discussed in [54], data growth and changes in the input make BaaS complicated. Workloads are almost never repeated exactly, as the input changes between executions, requiring the BaaS provider to monitor and react to changes. Moreover, cloud providers change machine types, parameters, and pricing very frequently — e.g., between 2012 and 2015, AWS introduced on average 2.6 new instances every three months. When these settings change, resource selection needs to be re-evaluated, even if a workload stays the same.

4.7.4.2 VCG-auction-based Approach

In this experiment, we compare our model with an VCG auction model. In a strategy-proof VCG auction, the agents truthfully reveal their best costs of executing a task. Then the consumer selects the agent with the best utility but pays according to the second best utility. Therefore, given a specific task, only the best and the second best agents' contract together define the price. So we create one agent who is always able to find the best configuration for each task, and another agent who is doing just worse than the best one. We assume a delta Δ that is the difference between the best utility and the second best utility, and vary this delta to see how much profit the best agent can get.

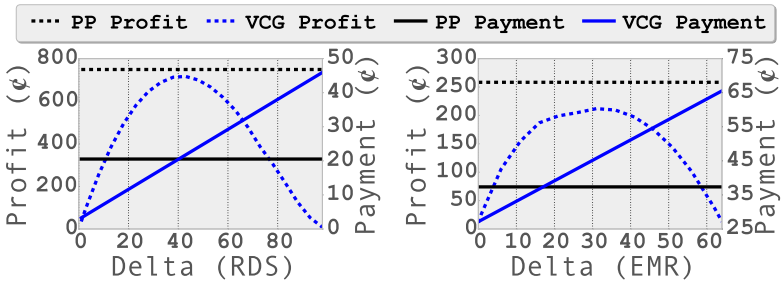


Figure 4.14: VCG auction brings less profit to agents without necessarily reducing consumers' payments. (PP = our posted-price approach)

As depicted in Figure 4.14, when Δ becomes greater, the best agent's profit goes up but then drops down. This is because of the demand. Larger Δ means more profit of each contract but less demand. So the profit reaches maximum at a certain point. Even this maximum VCG profit is less than the profit in our approach. It is because our approach optimizes the profit for each individual task in a workload, while the VCG approach in this experiment applies unified profit. Moreover, when Δ gets greater, consumers' average payment for each contract is bigger. This is by definition of VCG. The VCG's average payment goes higher than our approach's when Δ is very small. In a word, our approach brings more profit to agents without necessarily increasing consumer's payment.

4.8 Related Work

In contrast to our market framework, which emphasizes the consumer need for task-level pricing, existing work on cloud pricing largely focuses on resource usage. One study used game theory to model a pricing framework where consumers compete with each other to maximize their utilities [7, 64]. Specifically, each consumer has a demand on resources, and their utility is a function of demand and price. A choice of price by a service provider triggers a change in the consumers' demands to maximize their utilities, thus affecting the provider's revenue. This work makes two key assumptions that are not present in our framework. First, the chosen utility functions indicate that the quality of service (QoS) de-

grades when consumers share resources. While meaningful for resources such as wireless bandwidth, this assumption has been shown to not always hold in many types of resources relevant to computation [4, 5]. In fact, QoS can improve when, for example, consumers share data and cache, and agents in our framework can take advantage of this to make more profit. Second, it is assumed that the consumers know each other's demands and strategies, and adjust their demands accordingly. In contrast, in our framework we consider consumers' tasks separately and use probability distributions to model runtime and financial cost, leading to a simpler yet practical model.

Variants of pricing mechanisms assume that providers price dynamically, based on the consumer arrival and departure rates [9, 117, 173, 174, 175]. In turn, prices also guide consumer demand. In a different direction, Ibrahim et al. [80] argue that the interference across virtual machines sharing the same hardware leads to overcharging. They suggest cloud providers to price based on effective virtual machine time. This framework guarantees benefits to consumers and urges providers to improve their system design.

Wong et al. [84] have compared three different pricing strategies in terms of fairness and revenue: (1) Bundled pricing, in which providers sell resource bundles (e.g., virtual machine with CPU, memory, and other resources) to consumers; (2) Resource pricing, in which providers charge consumers separate prices for the consumed resources; (3) Differentiated pricing, in which providers charge consumers personalized prices. They define fairness as a function of equitability and efficiency of utilities among all consumers and conclude that differentiated pricing provides the best fairness. They treat consumers' jobs identically and define fairness based on the number of jobs that are successfully executed by the cloud provider. They do not consider the connection between uncertain completion time and utility.

Economic-based resource allocation has been extensively studied in grid computing [21, 22, 67, 130, 93]. Researchers have developed different economic models in two main categories: "commodity markets" and "auctions". In a commodity market, resources are

sold at a posted-price. The price of resources affects consumers' utility and demand, and therefore impacts the providers' profit. Finding the equilibrium is the main focus in these models. Yeo et al. proposed a utility-driven pricing function [179, 141] and an autonomic pricing approach [180]. Stuer et al. [144] adapted Smale's method [143] to price resources in grid computing markets. Bossenbroek et al. [18] introduced option contracts into the market and used hedge strategies to reduce consumers' risk of missing task deadlines. Auction-based pricing in grid computing contains several different forms. Double auction requires consumers and providers to publish their requests and offers in a marketplace [71, 148, 165, 81]. Vickrey auction is a type of sealed-bid auction in which the highest bidder wins but pays the second-highest bid [159, 122]. Combinatorial auction allows consumers to bid on combinations of resources [34].

Posted-price selling and auctions are both established ways of selling, and it is not clear which one is better. The key challenge is the uncertainty of the value of the commodity (in this case, the computational resource) and researchers have developed different models to compare the two mechanisms under various assumptions. Computer scientists measure system metrics in these two mechanisms. Wolski et al. [167] state that posted-price brings more price stability, higher task completion rate and higher resource utilization ratio than auctions. Vanmechelen and Broeckhove [154] conclude that posted-price results in more stable pricing, while Vickrey auction results in fewer message passing in dynamic pricing. Economists have discussed the revenues in these two mechanisms. Wang [161] compares posted-price selling and auctions where the buyers have independent private values of the commodity. He finds that posted-price selling brings more profit to the seller when the buyers' values of the commodity are widely dispersed. Campbell and Levin [23] state that auctions perform worse than posted-price when buyer valuations are interdependent. Hammond [65, 66] concludes that revenues of the two mechanisms cannot be statistically distinguished based on his study on eBay.

In contrast to our framework, this entire body of work focuses on resource-level pricing, and does not provide a mechanism for consumers to select resources based on their tasks. Recent work has started shifting the focus to task-level pricing. Floratou et al. [54] propose a Benchmark as a Service (BaaS) that benchmarks user's workload and suggests the optimal configuration for repetitive execution. As they mention in the paper, changes such as growth of input data make BaaS complicated. In our approach, we do not assume repetitiveness of workloads. Consumers may pay more for extremely repetitive workloads, but are free from benchmarking evolved workloads.

Auction-based models [150, 149] assume that providers bid for service contracts. These models use the Vickrey-Clarke-Groves auction mechanism, which redefines the payment to the winner and guarantees that all providers report their true cost of providing the service. While this work provides a good model for task-level pricing, it does not consider execution time for tasks. In our framework, we balance the consumers' trade-off of execution time and price through their utility function.

Personalized Service Level Agreements (PSLA) [124, 125, 126, 127] resemble the contracts in our framework, and describe a vision of a system that analyzes consumers' data and suggests to them tiers of service. Each tier describes three properties: completion time, price per hour, querying capabilities. For example, a tier on Amazon EMR can be (< 3.5 minutes, \$0.12/hour, SELECT 1 attribute FROM 1 table WHERE condition). In our framework, consumers do not subscribe to a tier of service, but rather provide the task they need and the agent provides a specific price for the task.

When multiple agents find the same best configuration for some tasks, their prices affect each other and finally converge to the Nash Equilibrium in differentiated Bertrand model [136] in the long run.

The agents in our computation framework derive estimates of cost and time. Several approaches employ machine learning to predict the execution time of a query [57, 6]. Li et al. [101] use statistical model to estimate CPU time and other resource comsumption. Duggan

et al. [48, 49] introduce special metrics and predict performance based on sampling. Wu et al. analyze the query execution plan directly to derive runtime predictions [170, 169], or use probabilistic models [171]. Ye et al. [178] perform service composition given the resource requirement of individual tasks. Uncertainty of time and cost is an important component in our framework. Existing work on scheduling SLAs considers uncertainty in the completion time when contracts specify a price. Specifically, scheduling considers 3 possible outcomes: (1) the provider accepts the SLA and returns results before the deadline, earning some profit; (2) the provider accepts the SLA but misses the deadline, and pays some penalty to the consumer; (3) the provider rejects the SLA and pays some penalty immediately. Xiong et al. [172] have developed an SLA admission control system that predicts the distribution of completion time, and accepts or rejects SLAs based on the expected profit. Chi et al. [30] assume a stream of SLAs all of which must be accepted. Their system minimizes loss by determining the execution order of SLAs based on the uncertain completion time and the penalty of missing the deadline. Liu et al. [108] have proposed an algorithm to solve tenant placement in the cloud given the distribution of completion time and the penalty of SLAs. Our market works differently in two aspects. First, our contracts consist of multiple target times, which are more flexible than the single deadline implicit in these SLAs. Second, we do not require the consumer to propose an SLA that may be rejected. Instead, the consumer makes a request that is priced by the agent according to their capabilities.

Fine-grained contract pricing is related to query optimization in distributed databases [98, 58] as we execute subtasks using different virtual machines. However, contract optimization has two objectives (time and cost), while query optimization has only one (time). These two objectives propagate differently in the task graph, making the problem more difficult.

4.9 Discussion

Our framework can be easily extended to handle applications with different QoS parameters. For example, in long-running services, completion time is not relevant and thus should not be part of the utility function. In contrast, other factors, such as response time, are important. These parameters are also uncertain due to unstable cloud performance [110, 162]. While we did not experiment with alternative QoS parameters and different application settings, our market framework is already equipped to handle them with appropriate changes to the utility function.

A meaningful extension to our work is to augment the market to handle varying prices. Our current framework assumes fixed prices for resource configurations. However, fluctuating prices do exist in the real world. For example, Amazon EC2 allows agents bid spot instances with much lower price [182, 3]. Agents set a maximum price threshold when requesting a spot instance. The request can be fulfilled when the market price of a spot instance is lower than this threshold. If the market price increases above the threshold, the spot instance will be terminated. In addition, agents can rent reserved instances either directly from Amazon EC2 or through its Reserved Instance Marketplace. In these cases, agents have more options to execute a task: 1) buy spot instances; 2) use their previously reserved instances; 3) buy reserved instances from others. These options introduce two additional factors to the market. First, the market needs to account for a supply function $S(\alpha_C, t)$. This means there are $S(\alpha_C, t)$ instances with rate α_C and available time t, where α_C is usually lower than the regular instance rate and t must be a limited period ranging from hours to years. Second, the framework needs to consider the starting time of a task. The starting time is inconsequential when there are only regular instances with fixed rates: The agent starts regular instances whenever a consumer accepts the contract. However, the starting time matters when the rates fluctuate. In this case, agents need to estimate (a) the supply function at different points in time to ensure enough machine hours for finishing consumers' tasks, and (b) the demand function at different points in time to decide how

many instances they want to reserve. This is not a straightforward extension to our work, and will likely lead to a more complex market model.

CHAPTER 5

CONCLUSIONS

This thesis proposes several new technologies to enhance the ease of use of databases in two dimensions: we help users process democratized data and exploit democratized computational capabilities.

In Chapter 2, we study the problem of synthesizing mapping relationships using tables. Mapping relationships, such as `(country, country-code)` or `(company, stock-ticker)`, are versatile data assets for an array of applications in data cleaning and data integration like auto-correction and auto-join. Our synthesis process leverages compatibility of tables based on co-occurrence statistics, as well as constraints such as functional dependency. Experiment results using web tables and enterprise spreadsheets suggest that the proposed approach can produce high quality mappings. Our work is a first step in the direction to facilitate the curation of mapping relationships. Questions that we would like to address in the future include: (1) how to best present related result clusters with overlapping values to human users to solicit feedback, so that users will not be confused by clusters with repeating values; (2) how to complement the corpus-driven approach to better cover mappings with large numbers of instances, by using other sources such as authoritative third-party data sets.

In Chapter 3, we study the query result diversification problem. We propose RC-Index to significantly reduce the number of items we extract to answer a range query. Compared to the state-of-the-art algorithms which are linear or quadratic, our query time is sublinear with respect to the number of items that satisfy the query. Moreover, the quality of our result has an approximation ratio. It is tunable through two parameters. When the

parameters are large, the approximation ratio approaches $1/2$, which is the best possible ratio of a polynomial algorithm unless P=NP. We experiment with synthetic data and real data. The result shows the quality of our result is close to or even better than the `Greedy` algorithm whose approximation ratio is $1/2$. In addition, our RC-Index is efficient to create. We can index 10^6 items within 6 minutes.

In Chapter 4, we propose a new marketplace framework that consumers can use to pay for well-defined database computations in the cloud. In contrast with existing pricing mechanisms, which are largely resource-centric, our framework introduces agent services that can leverage a plethora of existing tools for time, cost estimation, and scheduling, to provide consumers with personalized cloud-pricing contracts targeting a specific computational task. Agents price contracts to maximize the utility offered to consumers while also producing a profit for their services. Our market can operate in conjunction with existing cloud markets, as it does not alter the way cloud providers offer and price services. It simplifies cloud use for consumers by allowing them to compare contracts, rather than choose resources directly. The market also allows users to extract more value from public cloud resources, achieving cheaper and faster query processing than naïve configurations, while a portion of this value is earned by the agents as profit for their services. Our experimental evaluation using the AWS cloud computing platform demonstrated that our market framework offers incentives to consumers, who can execute their tasks more cost-effectively, and to agents, who make profit from providing fair and competitive contracts.

In summary, this thesis has studied how to enhance database usability in data exploration. With the help of the techniques we have proposed, users can perform data transformation using materialized mapping relationships conveniently, explore the data by querying diverse and representative items efficiently, and configure the database easily in a market.

BIBLIOGRAPHY

[1] Abedjan, Ziawasch, Morcos, John, Gubanov, Michael N., Ilyas, Ihab F., Stonebraker, Michael, Papotti, Paolo, and Ouzzani, Mourad. Dataxformer: Leveraging the web for semantic transformations. In *CIDR* (2015).

[2] Aboulnaga, Ashraf, Wang, Ziyu, and Zhang, Zi Ye. Packing the most onto your cloud. In *CloudDB* (2009), pp. 25–28.

[3] Agmon Ben-Yehuda, Orna, Ben-Yehuda, Muli, Schuster, Assaf, and Tsafrir, Dan. Deconstructing amazon ec2 spot instance pricing. *TEAC 1*, 3 (2013), 16:1–16:20.

[4] Ahmad, Mumtaz, Aboulnaga, Ashraf, Babu, Shivnath, and Munagala, Kamesh. Interaction-aware scheduling of report-generation workloads. *VLDBJ 20*, 4 (2011), 589–615.

[5] Ahmad, Mumtaz, Duan, Songyun, Aboulnaga, Ashraf, and Babu, Shivnath. Predicting completion times of batch query workloads using interaction-aware models and simulation. In *EDBT/ICDT* (2011), pp. 449–460.

[6] Akdere, M., Çetintemel, U., Riondato, M., Upfal, E., and Zdonik, S.B. Learning-based query performance modeling and prediction. In *ICDE* (2012), pp. 390–401.

[7] Al Daoud, Ashraf, Agarwal, Sachin, and Alpcan, Tansu. Brief announcement: Cloud computing games: Pricing services of large data centers. In *DISC* (2009), pp. 309–310.

[8] Amer-Yahia, Sihem, Cho, SungRan, and Srivastava, Divesh. Tree pattern relaxation. In *EDBT* (2002), pp. 496–513.

[9] An, Bo, Lesser, Victor, Irwin, David, and Zink, Michael. Automated negotiation with decommitment for dynamic resource allocation in cloud computing. In *AAMAS* (2010), pp. 981–988.

[10] Bejerano, Yigal, Smith, Mark A., Naor, Joseph, and Immorlica, Nicole. Efficient location area planning for personal communication systems. In *Transaction of Networking* (2006).

[11] Bentley, Jon Louis. Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9 (Sept. 1975), 509–517.

[12] Bentley, Jon Louis. Decomposable searching problems. *Inf. Process. Lett. 8*, 5 (1979), 244–251.

[13] Bentley, Jon Louis, and Friedman, Jerome H. Data structures for range searching. *ACM Comput. Surv. 11*, 4 (1979), 397–409.

[14] Bernstein, Philip A., Madhavan, Jayant, and Rahm, Erhard. Generic schema matching, ten years later. In *Proceedings of VLDB* (2011).

[15] Beygelzimer, Alina, Kakade, Sham, and Langford, John. Cover trees for nearest neighbor. In *ICML* (2006), pp. 97–104.

[16] Blackard, Jock A., and Dean, Denis J. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture 24*, 3 (1999), 131 – 151.

[17] Bollacker, Kurt, Evans, Colin, Paritosh, Praveen, Sturge, Tim, and Taylor, Jamie. Freebase: A collaboratively created graph database for structuring human knowledge. In *SIGMOD* (2008), pp. 1247–1250.

[18] Bossenbroek, A., Tirado-Ramos, A., and Sloot, P.M.A. Grid resource allocation by means of option contracts. *ISJ 3*, 1 (2009), 49–64.

[19] Broder, A. On the resemblance and containment of documents. In *SEQUENCES* (1997), IEEE Computer Society, pp. 21–.

[20] Bronzi, Mirko, Crescenzi, Valter, Merialdo, Paolo, and Papotti, Paolo. Extraction and integration of partially overlapping web sources. *PVLDB* (2013), 805–816.

[21] Buyya, R., Abramson, D., and Venugopal, S. The grid economy. *IEEE 93*, 3 (2005), 698–714.

[22] Buyya, Rajkumar. Market-oriented cloud computing: Vision, hype, and reality of delivering computing as the 5th utility. In *CCGRID* (2009), pp. 1–1.

[23] Campbell, Colin M., and Levin, Dan. When and why not to auction. *Econ. Theory 27*, 3 (2006), 583–596.

[24] Carbonell, Jaime, and Goldstein, Jade. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR* (1998), pp. 335–336.

[25] Ceccarello, Matteo, Pietracaprina, Andrea, Pucci, Geppino, and Upfal, Eli. Mapreduce and streaming algorithms for diversity maximization in metric spaces of bounded doubling dimension. *Proc. VLDB Endow. 10*, 5 (2017), 469–480.

[26] Chakrabarti, Kaushik, Chaudhuri, Surajit, Chen, Zhimin, Ganjam, Kris, and He, Yeye. Data services leveraging bing's data assets. *IEEE Data Eng. Bull.* (2016).

[27] Chaudhuri, Surajit, Ramakrishnan, Raghu, and Weikum, Gerhard. Integrating db and ir technologies: What is the sound of one hand clapping. In *CIDR* (2005), pp. 1–12.

[28] Chen, Yang, Goldberg, Sean, Wang, Daisy Zhe, and Johri, Soumitra Siddharth. Ontological pathfinding: Mining first-order knowledge from large knowledge bases. In *SIGMOD* (2016).

[29] Chen, Yi, Wang, Wei, and Liu, Ziyang. Keyword-based search and exploration on databases. In *ICDE* (2011), pp. 1380–1383.

[30] Chi, Yun, Hacígümüş, Hakan, Hsiung, Wang-Pin, and Naughton, Jeffrey F. Distribution-based query scheduling. *PVLDB 6*, 9 (2013), 673–684.

[31] Chierichetti, Flavio, Dalvi, Nilesh, and Kumar, Ravi. Correlation clustering in mapreduce. In *KDD* (2014).

[32] Chitnis, Laukik, Das Sarma, Anish, Machanavajjhala, Ashwin, and Rastogi, Vibhor. Finding connected components in map-reduce in logarithmic rounds. In *ICDE* (2013), pp. 50–61.

[33] Choobineh, Joobin, Mannino, Michael V., and Tseng, Veronica P. A form-based approach for database analysis and design. *Commun. ACM 35*, 2 (Feb. 1992), 108–120.

[34] Chun, B.N., Buonadonna, P., AuYoung, A., Ng, Chaki, Parkes, D.C., Shneidman, J., Snoeren, A.C., and Vahdat, A. Mirage: A microeconomic resource allocation system for sensornet testbeds. In *EmNetS* (2005), pp. 19–28.

[35] Church, Kenneth Ward, and Hanks, Patrick. Word association norms, mutual information, and lexicography. *Comput. Linguist. 16*, 1 (1990), 22–29.

[36] Comer, Douglas. Ubiquitous b-tree. *ACM Comput. Surv. 11*, 2 (June 1979), 121–137.

[37] Cortez, Eli, Bernstein, Philip A., He, Yeye, and Novik, Lev. Annotating database schemas to help enterprise search. *Proceedings of VLDB* (2015).

[38] Dahlhaus, E., Johnson, D. S., Papadimitriou, C. H., Seymour, P. D., and Yannakakis, M. The complexity of multiterminal cuts. *SIAM J. Comput. 23*, 4 (1994), 864–894.

[39] Dar, Shaul, Entin, Gadi, Geva, Shai, and Palmon, Eran. Dtl's dataspot: Database exploration using plain language. In *VLDB* (1998), pp. 645–649.

[40] De Roure, David, Goble, Carole, and Stevens, Robert. The design and realisation of the virtual research environment for social sharing of workflows. *FGCS 25*, 5 (2009), 561–567.

[41] Demaine, Erik D., Emanuel, Dotan, Fiat, Amos, and Immorlica, Nicole. Correlation clustering in general weighted graphs. *Theor. Comput. Sci. 361*, 2 (2006), 172–187.

[42] Deng, Ting, and Fan, Wenfei. On the complexity of query result diversification. *Proc. VLDB Endow. 6*, 8 (2013), 577–588.

[43] Deng, Ting, and Fan, Wenfei. On the complexity of query result diversification. *ACM Trans. Database Syst. 39*, 2 (2014), 15:1–15:46.

[44] Drosou, M., and Pitoura, E. Diverse set selection over dynamic data. *IEEE Transactions on Knowledge and Data Engineering 26*, 5 (2014), 1102–1116.

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

[46] Drosou, Marina, and Pitoura, Evaggelia. Disc diversity: Result diversification based on dissimilarity and coverage. *Proc. VLDB Endow. 6*, 1 (2012), 13–24.

[47] Drosou, Marina, and Pitoura, Evaggelia. Dynamic diversification of continuous data. In *EDBT* (2012), pp. 216–227.

[48] Duggan, J., Chi, Yun, Hacigumus, H., Zhu, Shenghuo, and Çetintemel, U. Packing light: Portable workload performance prediction for the cloud. In *ICDEW* (2013), pp. 258–265.

[49] Duggan, Jennie, Papaemmanouil, Olga, Çetintemel, Ugur, and Upfal, Eli. Contender: A resource modeling approach for concurrent query performance prediction. In *EDBT* (2014), pp. 109–120.

[50] Embley, David W. Nfql: The natural forms query language. *ACM Trans. Database Syst. 14*, 2 (June 1989), 168–211.

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

[52] Farley, Benjamin, Juels, Ari, Varadarajan, Venkatanathan, Ristenpart, Thomas, Bowers, Kevin D., and Swift, Michael M. More for your money: Exploiting performance heterogeneity in public clouds. In *SOCC* (2012), pp. 20:1–20:14.

[53] Finkel, R. A., and Bentley, J. L. Quad trees a data structure for retrieval on composite keys. *Acta Inf. 4*, 1 (Mar. 1974), 1–9.

[54] Floratou, Avrilia, Patel, Jignesh M., Lang, Willis, and Halverson, Alan. When free is not really free: What does it cost to run a database workload in the cloud? In *TPCTC* (2012), pp. 163–179.

[55] Galárraga, Luis Antonio, Teflioudi, Christina, Hose, Katja, and Suchanek, Fabian. Amie: Association rule mining under incomplete evidence in ontological knowledge bases. In *WWW* (2013).

[56] Galhardas, Helena, Florescu, Daniela, Shasha, Dennis, and Simon, Eric. Ajax: An extensible data cleaning tool. In *SIGMOD* (2000), pp. 590–.

[57] Ganapathi, Archana, Kuno, Harumi, Dayal, Umeshwar, Wiener, Janet L., Fox, Armando, Jordan, Michael, and Patterson, David. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *ICDE* (2009), pp. 592–603.

[58] Ganguly, Sumit, Hasan, Waqar, and Krishnamurthy, Ravi. Query optimization for parallel execution. In *SIGMOD* (1992), pp. 9–18.

[59] Garg, Naveen, Vazirani, Vijay V, and Yannakakis, Mihalis. Multiway cuts in directed and node weighted graphs. In *ICALP* (1994), pp. 487–498.

[60] Gollapudi, Sreenivas, and Sharma, Aneesh. An axiomatic approach for result diversification. In *WWW* (2009), pp. 381–390.

[61] Google Web Tables. `http://research.google.com/tables`.

[62] Gruenheid, Anja, Dong, Xin Luna, and Srivastava, Divesh. Incremental record linkage. *Proc. VLDB Endow. 7*, 9 (May 2014), 697–708.

[63] Gupta, Rahul, Halevy, Alon, Wang, Xuezhi, Whang, Steven Euijong, and Wu, Fei. Biperpedia: An ontology for search applications. *PVLDB* (2014), 505–516.

[64] Hadji, Makhlouf, Louati, Wajdi, and Zeghlache, Djamal. Constrained pricing for cloud resource allocation. In *NCA* (2011), pp. 359–365.

[65] Hammond, Robert G. Comparing revenue from auctions and posted prices. *IJIO 28*, 1 (2010), 1–9.

[66] Hammond, Robert G. A structural model of competing sellers: Auctions and posted prices. *Eur. Econ. Rev. 60*, C (2013), 52–68.

[67] Haque, Aminul, Alhashmi, Saadat M., and Parthiban, Rajendran. A survey of economic models in grid computing. *FGCS 27*, 8 (2011), 1056–1069.

[68] He, Bin, and Chang, Kevin Chen-Chuan. Statistical schema matching across web query interfaces. In *Proceedings of SIGMOD* (2003).

[69] He, Hai, Meng, Weiyi, Yu, Clement, and Wu, Zonghuan. Wise-integrator: An automatic integrator of web search interfaces for e-commerce. *VLDB Journal* (2004).

[70] He, Hai, Meng, Weiyi, Yu, Clement T., and Wu, Zonghuan. Wise-integrator: An automatic integrator of web search interfaces for e-commerce. In *PVLDB* (2003).

[71] He, Minghua, Leung, Ho-fung, and Jennings, Nicholas R. A fuzzy-logic based bidding strategy for autonomous agents in continuous double auctions. *TKDE 15*, 6 (2003), 1345–1363.

[72] He, Yeye, Ganjam, Kris, and Chu, Xu. Sema-join: joining semantically-related tables using big table corpora. In *Proceedings of VLDB* (2015).

[73] Hellerstein, Joseph M., Naughton, Jeffrey F., and Pfeffer, Avi. Generalized search trees for database systems. In *VLDB* (1995), pp. 562–573.

[74] Herodotou, Herodotos, and Babu, Shivnath. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. *PVLDB 4*, 11 (2011), 1111–1122.

[75] Herodotou, Herodotos, Dong, Fei, and Babu, Shivnath. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In *SOCC* (2011), pp. 18:1–18:14.

[76] Hopcroft, J. E., and Ullman, J. D. Set merging algorithms. *SIAM Journal on Computing 2*, 4 (1973), 294–303.

[77] Hu, T. C. Multi-commodity network flows. *Operations Research 11*, 3 (1963), 344–360.

[78] Huerta, Ramon, Mosqueiro, Thiago, Fonollosa, Jordi, Rulkov, Nikolai F, and Rodriguez-Lujan, Irene. Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring. *Chemometrics and Intelligent Laboratory Systems 157* (2016), 169 – 176.

[79] Huhtala, Ykä, Kärkkäinen, Juha, Porkka, Pasi, and Toivonen, Hannu. Tane: An efficient algorithm for discovering functional and approximate dependencies. *Computer Journal 42*, 2 (1999).

[80] Ibrahim, Shadi, He, Bingsheng, and Jin, Hai. Towards pay-as-you-consume cloud computing. In *SCC* (2011), pp. 370–377.

[81] Izakian, Hesam, Abraham, Ajith, and Ladani, Behrouz Tork. An auction method for resource allocation in computational grids. *FGCS 26*, 2 (2010), 228–235.

[82] Jacob, Joseph C., Katz, Daniel S., Berriman, G. Bruce, Good, John C., Laity, Anastasia C., Deelman, Ewa, Kesselman, Carl, Singh, Gurmeet, Su, Mei-Hui, Prince, Thomas A., and Williams, Roy. Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. *IJCSE 4*, 2 (2009), 73–87.

[83] Jagadish, H. V., Chapman, Adriane, Elkiss, Aaron, Jayapandian, Magesh, Li, Yunyao, Nandi, Arnab, and Yu, Cong. Making database systems usable. In *SIGMOD* (2007), pp. 13–24.

[84] Joe-Wong, Carlee, and Sen, Soumya. Mathematical frameworks for pricing in the cloud: Revenue, fairness, and resource allocations. *CoRR abs/1212.0022* (2012).

[85] Juve, Gideon, Chervenak, Ann, Deelman, Ewa, Bharathi, Shishir, Mehta, Gaurang, and Vahi, Karan. Characterizing and profiling scientific workflows. *FGCS 29*, 3 (2013), 682–692.

[86] Kandel, Sean, Paepcke, Andreas, Hellerstein, Joseph, and Heer, Jeffrey. Wrangler: Interactive visual specification of data transformation scripts. In *CHI* (2011), pp. 3363–3372.

[87] Karampaglis, Zisis, Gounaris, Anastasios, and Manolopoulos, Yannis. A bi-objective cost model for database queries in a multi-cloud environment. In *MEDES* (2014).

[88] Karger, David R., and Ruhl, Matthias. Finding nearest neighbors in growth-restricted metrics. In *STOC* (2002), pp. 741–750.

[89] Khan, H. A., and Sharaf, M. A. Progressive diversification for column-based data exploration platforms. In *ICDE* (2015), pp. 327–338.

[90] Khan, Hina A., Drosou, Marina, and Sharaf, Mohamed A. Dos: An efficient scheme for the diversification of multiple search results. In *SSDBM* (2013), pp. 40:1–40:4.

[91] Khan, Hina A., Drosou, Marina, and Sharaf, Mohamed A. Scalable diversification of multiple search results. In *CIKM* (2013), pp. 775–780.

[92] Khan, Hina A., Sharaf, Mohamed A., and Albarrak, Abdullah. Divide: Efficient diversification for interactive data exploration. In *SSDBM* (2014), pp. 15:1–15:12.

[93] Kiefer, Tim, Schön, Hendrik, Habich, Dirk, and Lehner, Wolfgang. A query, a minute: Evaluating performance isolation in cloud databases. In *TPCTC* (2014), pp. 173–187.

[94] Kimball, Ralph, and Ross, Margy. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. John Wiley & Sons, Inc., 2002.

[95] Kokkinos, P., Varvarigou, T. A., Kretsis, A., Soumplis, P., and Varvarigos, E. A. Cost and utilization optimization of amazon ec2 instances. In *CLOUD* (2013), pp. 518–525.

[96] Koudas, Nick, Sarawagi, Sunita, and Srivastava, Divesh. Record linkage: Similarity measures and algorithms. In *SIGMOD* (2006), pp. 802–803.

[97] Krauthgamer, Robert, and Lee, James R. Navigating nets: Simple algorithms for proximity search. In *SODA* (2004), pp. 798–807.

[98] Lanzelotte, Rosana S. G., Valduriez, Patrick, and Zaït, Mohamed. On the effectiveness of optimization search strategies for parallel execution spaces. In *VLDB* (1993), pp. 493–504.

[99] Lee, D. T., and Wong, C. K. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Inf. 9*, 1 (Mar. 1977), 23–29.

[100] Lee, D. T., and Wong, C. K. Quintary trees: A file structure for multidimensional datbase sytems. *ACM Trans. Database Syst. 5*, 3 (1980), 339–353.

[101] Li, Jiexing, König, Arnd Christian, Narasayya, Vivek, and Chaudhuri, Surajit. Robust estimation of resource consumption for sql queries using statistical techniques. *PVLDB 5*, 11 (2012), 1555–1566.

[102] Li, Xian, Dong, Xin Luna, Lyons, Kenneth, Meng, Weiyi, and Srivastava, Divesh. Truth finding on the deep web: is the problem solved? In *PVLDB* (2013), pp. 97–108.

[103] Li, Yaliang, Gao, Jing, Meng, Chuishi, Li, Qi, Su, Lu, Zhao, Bo, Fan, Wei, and Han, Jiawei. A survey on truth discovery. In *SIGKDD Exploration* (2016).

[104] Li, Yu, Lo, Eric, Yiu, Man Lung, and Xu, Wenjian. Query optimization over cloud data market. In *EDBT* (2015), pp. 229–240.

[105] Lichman, M. UCI machine learning repository, 2013.

[106] Lin, Thomas, Mausam, and Etzioni, Oren. Identifying functional relations in web text. In *Proceedings of EMNLP* (2010).

[107] Ling, Xiao, Halevy, Alon, Wu, Fei, and Yu, Cong. Synthesizing union tables from the web. In *IJCAI* (2013), pp. 2677–2683.

[108] Liu, Ziyang, Hacigümüş, Hakan, Moon, Hyun Jin, Chi, Yun, and Hsiung, Wang-Pin. Pmax: Tenant placement in multitenant databases for profit maximization. In *EDBT* (2013), pp. 442–453.

[109] Livny, Jonathan, Teonadi, Hidayat, Livny, Miron, and Waldor, Matthew K. High-throughput, kingdom-wide prediction and annotation of bacterial non-coding rnas. *PloS one 3*, 9 (2008), e3197.

[110] Lorenz, D.H., and Orda, A. Qos routing in networks with uncertain parameters. *TON 6*, 6 (1998), 768–778.

[111] Lueker, George S. A data structure for orthogonal range queries. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science* (1978), pp. 28–34.

[112] Lueker, George S., and Willard, Dan E. A data structure for dynamic range queries. *Information Processing Letters 15*, 5 (1982), 209 – 213.

[113] Malawski, Maciej, Juve, Gideon, Deelman, Ewa, and Nabrzyski, Jarek. Cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In *SC* (2012), pp. 22:1–22:11.

[114] Mankiw, N. Gregory. *Principles of Microeconomics, 7th Edition*, 7th ed. Cengage Learning, 2014.

[115] Marcus, Ryan, and Papaemmanouil, Olga. Wisedb: A learning-based workload management advisor for cloud databases. *Proc. VLDB Endow. 9*, 10 (2016), 780–791.

[116] Microsoft Excel Power Query. `http://office.microsoft.com/powerbi`.

[117] Mihailescu, M., and Teo, Yong Meng. Dynamic resource pricing on federated clouds. In *CCGrid* (2010), pp. 513–517.

[118] Minack, Enrico, Siberski, Wolf, and Nejdl, Wolfgang. Incremental diversification for very large sets: A streaming-based approach. In *SIGIR* (2011), pp. 585–594.

[119] Moro, Sérgio, Cortez, Paulo, and Rita, Paulo. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems 62* (2014), 22 – 31.

[120] Murray, Norman, Paton, Norman, and Goble, Carole. Kaleidoquery: A visual query language for object databases. In *AVI* (1998), pp. 247–257.

[121] Nievergelt, J., and Reingold, E. M. Binary search trees of bounded balance. *SIAM Journal on Computing 2*, 1 (1973), 33–43.

[122] Nisan, N., London, S., Regev, O., and Camiel, N. Globally distributed computation over the internet-the popcorn project. In *ICDCS* (1998), pp. 592–601.

[123] Omohundro, Stephen M. Five balltree construction algorithms. Tech. rep., 1989.

[124] Ortiz, Jennifer, de Almeida, Victor Teixeira, and Balazinska, Magdalena. A vision for personalized service level agreements in the cloud. In *DanaC* (2013), pp. 21–25.

[125] Ortiz, Jennifer, de Almeida, Victor Teixeira, and Balazinska, Magdalena. Changing the face of database cloud services with personalized service level agreements. In *CIDR* (2015).

[126] Ortiz, Jennifer, Lee, Brendan, and Balazinska, Magdalena. Perfenforce demonstration: Data analytics with performance guarantees. In *SIGMOD* (2016), pp. 2141–2144.

[127] Ortiz, Jennifer, Lee, Brendan, Balazinska, Magdalena, and Hellerstein, Joseph L. Perfenforce: A dynamic scaling engine for analytics with performance guarantees. *CoRR abs/1605.09753* (2016).

[128] Pandey, S., Barker, A., Gupta, K.K., and Buyya, R. Minimizing execution costs when using globally distributed cloud services. In *AINA* (2010), pp. 222–229.

[129] Qin, Lu, Yu, Jeffrey Xu, and Chang, Lijun. Diversifying top-k results. *Proc. VLDB Endow. 5*, 11 (July 2012), 1124–1135.

[130] Qureshi, Muhammad Bilal, Dehnavi, Maryam Mehri, Min-Allah, Nasro, Qureshi, Muhammad Shuaib, Hussain, Hameed, Rentifis, Ilias, Tziritas, Nikos, Loukopoulos, Thanasis, Khan, Samee U., Xu, Cheng-Zhong, and Zomaya, Albert Y. Survey on grid resource allocation mechanisms. *JGC 12*, 2 (2014), 399–441.

[131] Rahm, Erhard, and Bernstein, Philip A. A survey of approaches to automatic schema matching. *The VLDB Journal 10*, 4 (Dec. 2001), 334–350.

[132] Rahm, Erhard, and Bernstein, Philip A. A survey of approaches to automatic schema matching. *The VLDB Journal 10*, 4 (Dec. 2001), 334–350.

[133] Rahm, Erhard, and Do, Hong Hai. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull. 23*, 4 (2000), 3–13.

[134] Ramakrishnan, Lavanya, Chase, Jeffrey S., Gannon, Dennis, Nurmi, Daniel, and Wolski, Rich. Deadline-sensitive workflow orchestration without explicit resource control. *JPDC 71*, 3 (2011), 343–353.

[135] Raman, Vijayshankar, and Hellerstein, Joseph M. Potter's wheel: An interactive data cleaning system. In *VLDB* (2001), pp. 381–390.

[136] Rasmusen, Eric. *Games and information : an introduction to game theory*. Blackwell Pub., 2007.

[137] Ravi, S. S., Rosenkrantz, D. J., and Tayi, G. K. Heuristic and special case algorithms for dispersion problems. *Operations Research 42*, 2 (1994), 299–310.

[138] Ritter, Alan, Downey, Doug, Soderland, Stephen, and Etzioni, Oren. It's a contradiction—no, it's not: A case study using functional relations. In *EMNLP* (2008), pp. 11–20.

[139] Santos, Rodrygo L. T., Macdonald, Craig, and Ounis, Iadh. Search result diversification. *Found. Trends Inf. Retr. 9*, 1 (Mar. 2015), 1–90.

[140] Schad, Jörg, Dittrich, Jens, and Quiané-Ruiz, Jorge-Arnulfo. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *PVLDB 3*, 1-2 (2010), 460–471.

[141] Shin Yeo, Chee, and Buyya, Rajkumar. Pricing for utility-driven resource management and allocation in clusters. *IJHPCA 21*, 4 (2007), 405–418.

[142] Shneiderman, Ben. Improving the human factors aspect of database interactions. *ACM Trans. Database Syst. 3*, 4 (Dec. 1978), 417–439.

[143] Smale, Steve. A convergent process of price adjustment and global newton methods. *J. Math. Econ. 3*, 2 (1976), 107–120.

[144] Stuer, Gunther, Vanmechelen, Kurt, and Broeckhove, Jan. A commodity market algorithm for pricing substitutable grid resources. *FGCS 23*, 5 (2007), 688–701.

[145] Su, Weifeng, Wang, Jiying, and Lochovsky, Frederick. Holistic schema matching for web query interfaces. In *Proceedings of EDBT* (2006).

[146] Suchanek, Fabian M., Kasneci, Gjergji, and Weikum, Gerhard. Yago: A core of semantic knowledge. In *WWW* (2007), pp. 697–706.

[147] Tamir, Arie. Obnoxious facility location on graphs. *SIAM J. Discret. Math. 4*, 4 (Sept. 1991), 550–567.

[148] Tan, Zhu, and Gurd, John R. Market-based grid resource allocation using a stable continuous double auction. In *GRID* (2007), pp. 283–290.

[149] Tanaka, M., and Murakami, Y. Strategy-proof pricing for cloud service composition. *TCC PP*, 99 (2014), 1–1.

[150] Tanaka, Masahiro, and Murakami, Yohei. An efficient algorithm for strategy-proof service composition. In *SCC* (2013), pp. 105–112.

[151] Theobald, Martin, Schenkel, Ralf, and Weikum, Gerhard. An efficient and versatile query engine for topx search. In *VLDB* (2005), pp. 625–636.

[152] Tudoran, Radu, Costan, Alexandru, Antoniu, Gabriel, and Bougé, Luc. A performance evaluation of azure and nimbus clouds for scientific applications. In *CloudCP* (2012), pp. 4:1–4:6.

[153] Ukkonen, Esko. Algorithms for approximate string matching. *Inf. Control 64*, 1-3 (1985), 100–118.

[154] Vanmechelen, Kurt, and Broeckhove, Jan. A comparative analysis of single-unit vickrey auctions and commodity markets for realizing grid economies with dynamic pricing. In *GECON* (2007), pp. 98–111.

[155] Varian, H.R. *Intermediate Microeconomics: A Modern Approach*. W.W. Norton & Company, 2010.

[156] Varizani, Vijay. *Approximation algorithms*. Springer Verlag, 2001.

[157] Vee, Erik, Srivastava, Utkarsh, Shanmugasundaram, Jayavel, Bhat, Prashant, and Yahia, Sihem Amer. Efficient computation of diverse query results. In *ICDE* (2008), pp. 228–236.

[158] Vieira, Marcos R., Razente, Humberto L., Barioni, Maria C. N., Hadjieleftheriou, Marios, Srivastava, Divesh, Traina, Caetano, and Tsotras, Vassilis J. On query result diversification. In *ICDE* (2011), pp. 1163–1174.

[159] Waldspurger, C.A., Hogg, T., Huberman, B.A., Kephart, J.O., and Stornetta, W.S. Spawn: a distributed computational economy. *TSE 18*, 2 (1992), 103–117.

[160] Wang, Hongyi, Jing, Qingfeng, Chen, Rishan, He, Bingsheng, Qian, Zhengping, and Zhou, Lidong. Distributed systems meet economics: Pricing in the cloud. In *HotCloud* (2010).

[161] Wang, Ruqu. Auctions versus posted-price selling. *Am. Econ. Rev. 83*, 4 (1993).

[162] Wang, Shangguang, Zheng, Zibin, Sun, Qibo, Zou, Hua, and Yang, Fangchun. Cloud model for service selection. In *INFOCOM WKSHPS* (2011), pp. 666–671.

[163] Wang, Xiaolan, Dong, Xin Luna, and Meliou, Alexandra. Data x-ray: A diagnostic tool for data errors. In *SIGMOD* (2015), pp. 1231–1245.

[164] Weber, Roger, Schek, Hans-Jörg, and Blott, Stephen. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB* (1998), pp. 194–205.

[165] Wieczorek, Marek, Podlipnig, Stefan, Prodan, Radu, and Fahringer, Thomas. Applying double auctions for scheduling of workflows on the grid. In *SC* (2008), pp. 27:1–27:11.

[166] Willard, Dan E. *Predicate-Oriented Database Search Algorithms*. Outstanding Dissertations in the Computer Sciences. Garland Publishing, New York, 1978.

[167] Wolski, Rich, Plank, James S., Brevik, John, and Bryan, Todd. Analyzing market-based resource allocation strategies for the computational grid. *IJHPCA 15*, 3 (2001), 258–281.

[168] Wolstencroft, Katherine, Haines, Robert, Fellows, Donal, Williams, Alan R., Withers, David, Owen, Stuart, Soiland-Reyes, Stian, Dunlop, Ian, Nenadic, Aleksandra, Fisher, Paul, Bhagat, Jiten, Belhajjame, Khalid, Bacall, Finn, Hardisty, Alex, de la Hidalga, Abraham Nieva, Vargas, Maria P. Balcazar, Sufi, Shoaib, and Goble, Carole A. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *NAR 41*, Webserver-Issue (2013), 557–561.

[169] Wu, Wentao, Chi, Yun, Hacígümüş, Hakan, and Naughton, Jeffrey F. Towards predicting query execution time for concurrent and dynamic database workloads. *PVLDB 6*, 10 (2013), 925–936.

[170] Wu, Wentao, Chi, Yun, Zhu, Shenghuo, Tatemura, J., Hacigumus, H., and Naughton, J.F. Predicting query execution time: Are optimizer cost models really unusable? In *ICDE* (2013), pp. 1081–1092.

[171] Wu, Wentao, Wu, Xi, Hacigümüş, Hakan, and Naughton, Jeffrey F. Uncertainty aware query execution time prediction. *PVLDB 7*, 14 (2014), 1857–1868.

[172] Xiong, Pengcheng, Chi, Yun, Zhu, Shenghuo, Tatemura, Junichi, Pu, Calton, and HacigümüŞ, Hakan. Activesla: A profit-oriented admission control framework for database-as-a-service providers. In *SOCC* (2011), pp. 15:1–15:14.

[173] Xu, Hong, and Li, Baochun. Maximizing revenue with dynamic cloud pricing: The infinite horizon case. In *ICC* (2012), pp. 2929–2933.

[174] Xu, Hong, and Li, Baochun. Dynamic cloud pricing for revenue maximization. *TCC 1*, 2 (2013), 158–171.

[175] Xu, Hong, and Li, Baochun. A study of pricing for cloud resources. *PER 40*, 4 (2013), 3–12.

[176] Yakout, Mohamed, Ganjam, Kris, Chakrabarti, Kaushik, and Chaudhuri, Surajit. Infogather: Entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD* (2012).

[177] Yannakakis, Mihalis, Kanellakis, Paris C., Cosmadakis, Stavros S., and Papadimitriou, Christos H. Cutting and partitioning a graph after a fixed pattern. In *ICALP* (1983), Josep Diaz, Ed., pp. 712–722.

[178] Ye, Zhen, Bouguettaya, Athman, and Zhou, Xiaofang. Qos-aware cloud service composition based on economic models. In *ICSOC* (2012), pp. 111–126.

[179] Yeo, Chee Shin, Buyya, Rajkumar, Yeo, Chee Shin, and Buyya, Rajkumar. Pricing for utility-driven resource management and allocation in clusters. In *ADCOM* (2004), pp. 32–41.

[180] Yeo, Chee Shin, Venugopal, Srikumar, Chu, Xingchen, and Buyya, Rajkumar. Autonomic metered pricing for a utility computing service. *FGCS 26*, 8 (2010), 1368–1380.

[181] Yi, Sangho, Andrzejak, A., and Kondo, D. Monetary cost-aware checkpointing and migration on amazon cloud spot instances. *TSC 5*, 4 (2012), 512–524.

[182] Yi, Sangho, Kondo, D., and Andrzejak, A. Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In *CLOUD* (2010), pp. 236–243.

[183] Yu, Cong, Lakshmanan, Laks, and Amer-Yahia, Sihem. It takes variety to make a world: Diversification in recommender systems. In *EDBT* (2009), pp. 368–378.

[184] Yu, Jeffrey Xu, Qin, Lu, and Chang, Lijun. Keyword search in databases. *Synthesis Lectures on Data Management 1*, 1 (2009), 1–155.

[185] Zhang, Meihui, Hadjieleftheriou, Marios, Ooi, Beng Chin, Procopiuc, Cecilia M., and Srivastava, Divesh. Automatic discovery of attributes in relational databases. In *SIGMOD* (2011), pp. 109–120.

[186] Zheng, Kaiping, Wang, Hongzhi, Qi, Zhixin, Li, Jianzhong, and Gao, Hong. A survey of query result diversification. *Knowl. Inf. Syst. 51*, 1 (2017), 1–36.

[187] Zhou, A.C., and He, Bingsheng. Transformation-based monetary cost optimizations for workflows in the cloud. *TCC 2*, 1 (2014), 85–98.

[188] Zloof, Moshé M. Query-by-example: The invocation and definition of tables and forms. In *VLDB* (1975), pp. 1–24.