

University of Southampton Research Repository

Copyright © and Moral Rights for this thesis and, where applicable, any accompanying data are retained by the author and/or other copyright owners. A copy can be downloaded for personal non-commercial research or study, without prior permission or charge. This thesis and the accompanying data cannot be reproduced or quoted extensively from without first obtaining permission in writing from the copyright holder/s. The content of the thesis and accompanying research data (where applicable) must not be changed in any way or sold commercially in any format or medium without the formal permission of the copyright holder/s.

When referring to this thesis and any accompanying data, full bibliographic details must be given, e.g.

Thesis: Author (Year of Submission) "Full thesis title", University of Southampton, name of the University Faculty or School or Department, PhD Thesis, pagination.

Data: Author (Year) Title. URI [dataset]

UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

**Improving Data Markets: Enabling Diverse
Data Pricing Functions and Assisting
Buyers to Purchase Data within their
Budgets**

by

Mengya Liu

*A thesis for the degree of
Doctor of Philosophy*

June 2022

University of Southampton

Abstract

Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

Doctor of Philosophy

**Improving Data Markets: Enabling Diverse Data Pricing Functions and Assisting
Buyers to Purchase Data within their Budgets**

by Mengya Liu

Data has become a valuable commodity that is traded among data sellers and buyers on data markets. A data market cooperates with sellers to monetise their data while supporting the buyers to purchase data that meets their conditions. The existing studies of data markets offer good solutions to pricing buyers' requirements of data in the form of queries. Despite being considered a critical aspect of the data business, however, very few studies have been looked at the technical problem of meeting the financial wishes of sellers and buyers, i.e. the problem of enabling sellers' price data by diverse data pricing functions and assisting buyers to purchase data within their budgets.

This thesis reviews the existing models of data markets and posits that the existing marketplace-centred models hinder both buyers and sellers by failing to consider their expectations to purchase data within budget and utility constraints and autonomously pricing the data requirements, respectively. From the perspective of sellers, pricing their data, or claiming revenue for their data contributions, is a major aspect of their business. Current marketplaces use various practices to protect sellers from arbitrage, but letting a marketplace decide the data prices and revenues of sellers is not flexible enough for the sellers to manage data and control revenues. On the other hand, buyers expect marketplaces to make decisions for their benefit. For instance, when query answers can be derived from multiple sellers, and the sellers sell duplicated data at different prices, a marketplace can decide to purchase the duplicated data at lower prices. If marketplaces make decisions for buyers' benefits, data dealers will either end up with a decrease in the revenue of some sellers, or they will decrease the revenue of all sellers by splitting the payment equally. Both outcomes contravene sellers' expectations of maximising revenue. In addition, current markets tend to ignore the above expectation from buyers and the constraints they face, such as budget and data preferences.

We explore a method to remove these impediments and investigate the question of how a data market might enable sellers to set the data prices autonomously, while also assisting buyers to purchase data within their budgets. We analyse the problem from several

angles, including the setup of a data market and the subsequent optimisations required in trading data. We introduce a federated data market, Data Emporium, which allows sellers to price their data independently. Then, we focus on the challenge of finding the best purchase for buyers' money in Data Emporium: Given that sellers price their data sources using different methods, the ideal purchase is a subset of query answers, referred to as an allocation, that has maximum utility within the budget of buyers, especially when the price of the data depends on the number of data items returned or accessed in order to derive the query answer. We generalise the problem and explain its NP-hardness, and we especially study the new challenge in it when a class of pricing functions – access dependent pricing functions (ADPFs) – is used by sellers. ADPFs are a type of function that charge a different price for a set of data items than the total cost of the individual data items. Thereby, generating a set of data items to reduce the cost compared to their individual purchase, and thus the cost of the solutions derived from them becomes a new challenge. We introduce our two-step approach to solve the problem: a cost compression algorithm for minimising the cost of intermediate allocations, and heuristic algorithms, Greedy and 3DDP, to approximate the optimal allocation. We then present our experimental evaluation of their performance.

Regarding the conflict that a marketplace faces when it tries to satisfy the expectations of both sellers and buyers, we propose a new data market architecture, Free Market, as a further solution to the above-mentioned problem. The motivation for the design of Free Market derives from the following considerations: (1) when sellers doubt the equivalence of their income and contribution; (2) when buyers lose their interest in being victims to high bills and non-customised query answers to their constraints; and (3) when a single seller or a marketplace fails to answer a query, sellers and buyers may not trust that the marketplace is fair, and buyers face the challenge of purchasing data from multiple independent sellers or competing marketplaces. Therefore, we design a free market to let buyers and sellers exchange or share data directly without involving a third party and releasing the trust deposits. Distributed sellers autonomously manage and price their data and maintain a query executor for queries that can be as small as an individual trading their private data or as big as a union of data sources, such as an independent marketplace with its sellers. In the meantime, buyers have data requirements on the table and they keep the right to decide where their money goes and how to collect data. This design eases the arduous process of developing trust in data markets, as well as enabling sellers to receive data revenue from buyers immediately after providing data.

For the technical challenge of trading data in a free market with an awareness of the diverse data pricing functions as well as the budget and utility constraints of purchases, we let a marketplace announce the catalogue of data sources and use a local query engine to match the requirements of buyers and supplies of sellers. The announcement from a marketplace informs buyers of the information about the available data sources

in the market. It easily copes with the existing and leaving sellers. From the buyers' perspective, the local query engine seeks a method to execute queries over a huge number of available data sources in a data market while reducing the cost of query answers. At the same time, it allows buyers to set up constraints for query processing and supports buyers to purchase the desired answers within those constraints. This thesis exhibits our local query engine. We first use a minimum spanning tree to model the problem and then adjust the classic greedy algorithm into two solutions, Gen-Greedy and Sum-Greedy. Experiments conducted with thirty random pricing settings demonstrate that the two solutions can save 66% on costs compared to the state-of-the-art market, CostFed. Moreover, we demonstrate an approximation solution to plan query execution with a budget constraint.

This thesis provides a comparison of the general markets, data emporium and the free market with respect to their structures and services for sellers and buyers from the aspect of data management, trust, pricing, dependency, budget, efficiency, searching process and settlement.

Contents

List of Figures	xi
List of Tables	xv
Listings	xix
Declaration of Authorship	xxiii
Acknowledgements	xxv
1 Introduction	1
1.1 Research Problem	2
1.2 Research Challenges	5
1.3 Contributions	6
1.4 Thesis Structure	7
2 Literature Review	11
2.1 Architecture of Data Market	13
2.1.1 Participants in Data Trade	13
2.1.2 Structure of Data Market	14
2.2 Data Sources and Services in Data Market	15
2.2.1 Pricing Query Answers and Services	19
2.2.1.1 Negotiated Pricing Strategy	19
2.2.1.2 Multi-factor Functional Pricing Strategy	19
Basics	20
Desiderata	20
Formula	21
Share	22
2.3 Buyers' Requirements for Data	22
2.4 Data Searching in Data Markets	24
2.4.1 Federated Query Engine	26
2.4.1.1 Query Processing	26
2.4.1.2 Query Optimisation	27
2.5 Data Trade Deals	28
2.6 Performance Criteria	29
2.7 Existing Research Gap	30
2.8 Summary	31

3	Methodology	33
3.1	Research Questions	34
3.1.1	Research Question 1	34
3.1.2	Research Question 2	36
3.1.3	Research Question 3	39
3.2	Foundations	40
3.2.1	Format of Data and Query	40
3.2.2	Related Algorithms	40
3.3	Evaluation Design	41
3.4	Summary	42
4	Purchase Allocation for Limited Budgets within the Data Emporium	43
4.1	Data Emporium	43
4.1.1	Data Sources	44
4.1.2	Query Answer Derivation	44
	Running Example	45
4.1.3	Price of Data and Query Answers	45
4.2	Purchase Allocation	46
4.2.1	The PA Problem	46
4.2.2	ADPF	47
4.2.3	Complexity Analysis	48
4.3	Approach	49
4.3.1	Running Example	50
4.3.2	Cost Compression	50
4.3.3	Greedy Algorithm	52
4.3.4	3DDP Heuristic	53
4.4	Evaluation	56
4.4.1	Experiment Settings	57
	Data Sources Preparation	57
	Pricing Functions and Parameters	57
	Budget	58
	Utility	58
	Hardware and Metrics	58
4.4.2	Impact of Pricing Parameters	59
4.4.3	Impact of Utility Distribution	70
4.5	Summary	70
5	Free Market	73
5.1	Market Catalogue	74
5.1.1	Summary statistics of Data Sources	74
5.1.2	Access Interface	75
5.1.3	Pricing Strategy	76
5.2	Local Query Engine	76
5.3	Summary	78
6	Price-based Query Planning in the Free Market	79
6.1	Problem Statement	79

6.2	Price-Based Query Planning	81
6.2.1	Query Plan-Tree	81
6.2.2	Cost Estimation of Plan-Tree	83
6.2.2.1	General Cost Estimation	86
6.2.2.2	Summary-based Cost Estimation	87
6.2.3	Minimum Cost Spanning Plan-Tree	88
6.3	Evaluation	90
6.3.1	Experiment Setup	90
6.3.1.1	Data Sources	90
6.3.1.2	Query	91
6.3.1.3	Evaluation Metrics	92
6.3.2	Experiment Results	93
6.3.2.1	Cost-Efficiency	93
6.3.2.2	Impact of Query and Data	95
	CostFed vs Gen-Greedy	95
	CostFed vs Sum-Greedy	98
6.3.2.3	Price Sensitivity of Gen-Greedy and Sum-Greedy	100
6.3.2.4	Time	103
6.4	Summary	104
7	Price-based Query Planning within Budgets in the Free Market	105
7.1	Problem Description	105
7.2	Theoretical Framing	107
7.2.1	Weight	107
7.2.2	Cost Constraint	109
7.3	Minimum Runtime Plan-Tree within Budgets in the Free Market	110
7.3.1	Pruning Strategy	110
7.3.2	Approximation Strategy	111
7.3.2.1	Lagrangian Relaxation	111
7.3.2.2	Application and Implementation	113
7.4	Evaluation	115
7.4.0.1	Experiment Setting	115
7.4.0.2	Evaluation Metrics	116
7.4.1	Experiment Results	116
7.4.1.1	Costs and Sunk Costs	117
7.5	Summary	118
8	Market Comparison	121
8.1	Structure of Markets	121
8.2	Services for Sellers	123
8.2.1	Management	123
8.2.2	Trust	124
8.2.3	Pricing	125
8.2.4	Dependency	125
8.3	Service for Buyers	125
8.3.1	Budget	126
8.3.2	Efficiency	126

8.3.3	Searching Process	127
8.3.4	Settlement	128
8.4	Summary	128
9	Conclusion and Future Work	131
9.1	Future Work	133
Appendix A	Purchase Allocation Problem	137
Appendix A.1	Purchase Allocation Problem Formulation	137
Appendix A.2	Solution Size and Price Rate under Equal Utility Setting	139
Appendix A.3	Utility of Allocations under Sized-Distributions of Utilities Settings	143
Appendix B	Experiments Settings and Results of Evaluation in Chapter 6	155
Appendix B.1	Random Pricing Function Settings	155
Appendix B.2	Query Plans	157
Appendix B.3	Entire Experiment Results	301
Appendix B.3.1	Cost Efficiency	301
Appendix B.4	Entire Experiment Results	301
Appendix B.4.1	Cost	301
Appendix B.4.2	Time Efficiency	306
Appendix C	Illustration of Approximation Algorithm in Chapter 7	317
Appendix C.1	Compare Function of Approximation Algorithm	317
Appendix C.2	Analysis of Query Plan Enumeration	318
Appendix C.3	Query Plan Enumeration Algorithm	318
References		319

List of Figures

1.1	General Workflow of Data Market	2
1.2	Trust Options of Sellers	4
2.1	Architecture of Data Markets. Yellow components are the information accepted by a data marketplace. Red components and arrows are the process of trading data in a marketplace. Green components imply that the instantiation requires optimisation.	12
2.2	Processes of A Federated Query Engine Collecting Answer to A Query .	26
3.1	The problem of diverse pricing functions and buyer budget constraints within a federated data market.	35
3.2	Problem of Letting Buyers and Sellers Directly Exchange Data	37
4.1	Process for Solving the Example Problem Using Greedy Algorithm . . .	53
4.2	3DDP Algorithm Solving the Example Query within a Budget of 2.5 . .	55
4.3	Total Solutions in the Returned Allocations and Runtime of Brute Solution, Greedy and 3DDP (1)	60
4.4	Total Solutions in the Returned Allocations and Runtime of Brute Solution, Greedy and 3DDP (2)	61
4.5	(continued) The Price Rate of Allocations Generated for Query	62
4.6	Total Solutions And Price Rate of the Allocation of Greedy And 3DDP for Query LD5, LD6, LD8 over FlatPF[P] with Different P	64
4.7	Total Solutions And Price Rate of the Allocations of Greedy And 3DDP for Query LD5, LS6, LS7 over FMPF[P, N] with Different P	66
4.8	Total Utility of the Allocations Returned by Greedy and 3DDP for Query CD6 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities	67
4.9	Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD5 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities	68
4.10	Total Utility of the Allocations Returned by Greedy and 3DDP for Query LS6 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities	69
5.1	Architecture of the Free Market (The green lines represent the corresponding relationship between the records in the catalogue and the sellers in the market. The red lines illustrate the conducted access or data requirements from buyers to the selected sellers.)	74
5.2	Workflow of Local Query Engine	77

6.1	Query CD2 and Its Relevant Data Sources from FedBench	80
6.2	Complete Query Graph of John's Query	82
6.3	Spanning Plan-Trees Modelling the Two Query Plans of John's Query . .	83
6.4	Function for Cardinality Estimation of Joining An Intermediate Tree and A Triple Pattern Node	84
6.5	Progress of Incrementally Adding Triple Patterns into Query Plan-Trees in Figure 6.3	85
6.6	Cost Reduction Comparison over Random Pricing Function Settings . .	94
6.7	Stack Cost of Query Plans of Query LD8 in Group 4 of <i>Per</i> Experiment Class	96
6.8	Query Plans of CD6 with the Group 1 <i>Per</i> Pricing Function Setting . . .	97
6.9	CD6	97
6.10	Stack Cost of Query Plans of Query LD11 in Group 1 <i>Per</i> Experiment Class	98
6.11	Stack Costs of the Query Plans of Query CD5 in Group 6 of <i>Freemium</i> Experiment Class	99
6.12	Experiment Results of Query CD5 After Switching Triple Patterns	99
6.13	Stack Cost of Query Plans of Query CD6 Generated by Sum-Greedy in Group 2 of <i>Freemium</i> Experiment Class	100
6.14	Query Plan of LS5 Generated by CostFed	101
6.15	Query Plan Costs Generated by Gen-Greedy, Sum-Greedy and CostFed for Query LS5 in 6 Cost Ranking Cases	102
6.16	Average Running Time of the Three Algorithms for All Queries	104
7.1	Process of Generating a Plan-Tree of John's Query in Market C	108
7.2	A Spanning Tree and An Edge Intended to Remove	114
7.3	New Spanning Trees by Adding Edges	114
7.4	Query Plan for Query CD6 by the Prune Algorithm	118
7.5	Query Plan for Query CD6 by the Approximation Algorithm	118
8.1	Structure of a General Data Market	121
8.2	Legends of Market Structures	121
8.3	Structure of the Data Emporium	122
8.4	Structure of the Free Market	123
Appendix A.1	Total Solutions in the Returned Allocations and Price Rate of Greedy and 3DDP in Settings of <i>UBPF + Flat</i> (1)	139
Appendix A.2	Total Solutions in the Returned Allocations and Price Rate of Greedy and 3DDP in Settings of <i>UBPF + Flat</i> (2)	140
Appendix A.3	Total Solutions in the Returned Allocations and Price Rate of Greedy and 3DDP in Settings of <i>UBPF + Freemium</i> (1)	141
Appendix A.4	Total Solutions in the Returned Allocations and Price Rate of Greedy and 3DDP in Settings of <i>UBPF + Freemium</i> (2)	142
Appendix A.5	Total Utility of the Allocations Returned by Greedy and 3DDP for Query CD1 over <i>FlatPF[P]</i> and <i>FMPF[P, N]</i> with Different <i>P</i> under Sized-Distributions of Utilities	143
Appendix A.6	Total Utility of the Allocations Returned by Greedy and 3DDP for Query CD6 over <i>FlatPF[P]</i> and <i>FMPF[P, N]</i> with Different <i>P</i> under Sized-Distributions of Utilities	144

Appendix A.7 Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD1 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities	145
Appendix A.8 Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD2 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities	146
Appendix A.9 Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD3 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities	147
Appendix A.10 Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD4 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities	148
Appendix A.11 Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD5 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities	149
Appendix A.12 Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD6 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities	150
Appendix A.13 Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD8 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities	151
Appendix A.14 Total Utility of the Allocations Returned by Greedy and 3DDP for Query LS6 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities	152
Appendix A.15 Total Utility of the Allocations Returned by Greedy and 3DDP for Query LS7 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities	153

List of Tables

2.1	Literature Summary on the Research of Data Source	18
2.2	Summary of the Research on Data Requirements	23
2.3	<i>Dataset Marketplace</i> side literature	25
2.4	Evaluation Measurements for Welfare of Sellers and Buyers	30
3.1	Basic Pricing Functions. T denotes a set of data units, $o(T)$ is the number of units in T , P is the flat rate price of an entire data source, and R is the number of units free of charge.	42
4.1	Example Data Sources and Query Answers	44
4.2	Settings of the Budget (B), Range of Flat-rate Price (P) and Free Quota (N) of Pricing Functions for Queries	56
4.3	Sized-Distributions of Utilities	58
4.4	Comparison Metrics for Size	59
4.5	The Difference Comparison of Λ_P^D and Λ_P^G over Flat Price P and Runtime	63
4.6	Runtime of Experiment $UBPF + FlatPF$ and $UBPF + FMPF$	64
5.1	Example RDF Dataset	75
5.2	Summary Statistics of Example Dataset	75
6.1	Queries of FedBench And the Entire Number of the Results. #ER: the number of entire results with Data Sources ChEBI, DBpedia, DrugBank, KEGG, Geonames, Jamendo, LinkedMDB, NY Times, SW Dog Food Indexed as #1,#2,#3,#4,#5,#6,#7,#8,#9.	92
6.2	cost ranking cases	102
7.1	Examples for Possible Query Results and Costs in Different Cases	106
7.2	Costs of Query Plans of Query CD6 at Different Budgets	117
8.1	Market Comparison of Services for Sellers	124
8.2	Market Comparison of Services for Buyers	126
Appendix B.1	10 Random Settings of Freemium Pricing Function $f(X, L, p)$ for Data Sources	155
Appendix B.2	10 Random Settings of Prefixed Pricing Function $f(0, 0, p)$ for Data Sources	156
Appendix B.3	10 Random Settings of Flat Pricing Function $f(X, \infty, 0)$ for Data Sources	156
Appendix B.4	Query Plans of Query CD1 at Settings of 10 Flat Pricing Functions	157

Appendix B.5 Query Plans of Query CD1 at Settings of 10 freemium Pricing Functions	159
Appendix B.6 Query Plans of Query CD1 at Settings of 10 per Pricing Functions	162
Appendix B.7 Query Plans of Query CD2 at Settings of 10 Flat Pricing Functions	165
Appendix B.8 Query Plans of Query CD2 at Settings of 10 freemium Pricing Functions	167
Appendix B.9 Query Plans of Query CD2 at Settings of 10 per Pricing Functions	170
Appendix B.10 Query Plans of Query CD3 at Settings of 10 Flat Pricing Functions	173
Appendix B.11 Query Plans of Query CD3 at Settings of 10 freemium Pricing Functions	176
Appendix B.12 Query Plans of Query CD3 at Settings of 10 per Pricing Functions	179
Appendix B.13 Query Plans of Query CD4 at Settings of 10 Flat Pricing Functions	181
Appendix B.14 Query Plans of Query CD4 at Settings of 10 freemium Pricing Functions	184
Appendix B.15 Query Plans of Query CD4 at Settings of 10 per Pricing Functions	187
Appendix B.16 Query Plans of Query CD5 at Settings of 10 Flat Pricing Functions	190
Appendix B.17 Query Plans of Query CD5 at Settings of 10 freemium Pricing Functions	192
Appendix B.18 Query Plans of Query CD5 at Settings of 10 per Pricing Functions	195
Appendix B.19 Query Plans of Query CD6 at Settings of 10 Flat Pricing Functions	198
Appendix B.20 Query Plans of Query CD6 at Settings of 10 freemium Pricing Functions	201
Appendix B.21 Query Plans of Query CD6 at Settings of 10 per Pricing Functions	204
Appendix B.22 Query Plans of Query CD7 at Settings of 10 Flat Pricing Functions	206
Appendix B.23 Query Plans of Query CD7 at Settings of 10 freemium Pricing Functions	209
Appendix B.24 Query Plans of Query CD7 at Settings of 10 per Pricing Functions	212
Appendix B.25 Query Plans of Query LD6 at Settings of 10 Flat Pricing Functions	215
Appendix B.26 Query Plans of Query LD6 at Settings of 10 freemium Pricing Functions	217
Appendix B.27 Query Plans of Query LD6 at Settings of 10 per Pricing Functions	220
Appendix B.28 Query Plans of Query LD7 at Settings of 10 Flat Pricing Functions	222

Appendix B.29 Query Plans of Query LD7 at Settings of 10 freemium Pricing Functions	225
Appendix B.30 Query Plans of Query LD7 at Settings of 10 per Pricing Functions	228
Appendix B.31 Query Plans of Query LD8 at Settings of 10 Flat Pricing Functions	231
Appendix B.32 Query Plans of Query LD8 at Settings of 10 freemium Pricing Functions	233
Appendix B.33 Query Plans of Query LD8 at Settings of 10 per Pricing Functions	235
Appendix B.34 Query Plans of Query LD10 at Settings of 10 Flat Pricing Functions	238
Appendix B.35 Query Plans of Query LD10 at Settings of 10 freemium Pricing Functions	241
Appendix B.36 Query Plans of Query LD10 at Settings of 10 per Pricing Functions	244
Appendix B.37 Query Plans of Query LD11 at Settings of 10 Flat Pricing Functions	247
Appendix B.38 Query Plans of Query LD11 at Settings of 10 freemium Pricing Functions	249
Appendix B.39 Query Plans of Query LD11 at Settings of 10 per Pricing Functions	252
Appendix B.40 Query Plans of Query LS2 at Settings of 10 Flat Pricing Functions	255
Appendix B.41 Query Plans of Query LS2 at Settings of 10 freemium Pricing Functions	258
Appendix B.42 Query Plans of Query LS2 at Settings of 10 per Pricing Functions	261
Appendix B.43 Query Plans of Query LS3 at Settings of 10 Flat Pricing Functions	263
Appendix B.44 Query Plans of Query LS3 at Settings of 10 freemium Pricing Functions	266
Appendix B.45 Query Plans of Query LS3 at Settings of 10 per Pricing Functions	268
Appendix B.46 Query Plans of Query LS4 at Settings of 10 Flat Pricing Functions	269
Appendix B.47 Query Plans of Query LS4 at Settings of 10 freemium Pricing Functions	272
Appendix B.48 Query Plans of Query LS4 at Settings of 10 per Pricing Functions	275
Appendix B.49 Query Plans of Query LS5 at Settings of 10 Flat Pricing Functions	277
Appendix B.50 Query Plans of Query LS5 at Settings of 10 freemium Pricing Functions	279
Appendix B.51 Query Plans of Query LS5 at Settings of 10 per Pricing Functions	282
Appendix B.52 Query Plans of Query LS6 at Settings of 10 Flat Pricing Functions	285
Appendix B.53 Query Plans of Query LS6 at Settings of 10 freemium Pricing Functions	288
Appendix B.54 Query Plans of Query LS6 at Settings of 10 per Pricing Functions	291
Appendix B.55 Query Plans of Query LS7 at Settings of 10 Flat Pricing Functions	293
Appendix B.56 Query Plans of Query LS7 at Settings of 10 freemium Pricing Functions	295
Appendix B.57 Query Plans of Query LS7 at Settings of 10 per Pricing Functions	298
Appendix B.58 Experiment Results of Query Answer Cost with Settings of Flat	301
Appendix B.59 Experiment Results of Query Answer Cost with Settings of Freemium	303
Appendix B.60 Experiment Results of Query Answer Cost with Settings of Per	304

Appendix B.61 Experiment Runtime (msec) of Query Planning Optimization Algorithms with Settings of Flat. C: CostFed. G: Gen-Greedy. S: Sum-Greedy.	306
Appendix B.62 Experiment Runtime (msec) of Query Planning Optimization Algorithms with Settings of Freemium. C: CostFed. G: Gen-Greedy. S: Sum-Greedy.	308
Appendix B.63 Experiment Runtime (msec) of Query Planning Optimization Algorithms with Settings of Per. C: CostFed. G: Gen-Greedy. S: Sum-Greedy.	309
Appendix B.64 Experiment Results of Query Planning Time (msec) with Settings of Flat. C: CostFed. G: Gen-Greedy. S: Sum-Greedy.	311
Appendix B.65 Experiment Results of Query Planning Time (msec) with Settings of Freemium. C: CostFed. G: Gen-Greedy. S: Sum-Greedy.	313
Appendix B.66 Experiment Results of Query Planning Time (msec) with Settings of Per. C: CostFed. G: Gen-Greedy. S: Sum-Greedy.	315

Nomenclature

$\delta(tp)$	the set of variables in a triple pattern
$\mathcal{L}(tp)$	the set of selected data sources to request matching triples of the triple patterns
$\mathcal{O}(tp, d)$	the set of triples matching the triple patterns in the data source
\mathcal{T}	the spanning plan-tree
\mathcal{X}	the problem of price-based query planning within a budget
$\Pi(\mathcal{T})$	the array that tracks the cardinality of intermediate results of a spanning plan-tree
\tilde{N}	the set of triple pattern node in a spanning plan-tree
$amount(v, D)$	the amount of matched terms to the variable over data sources in D
B	the budget of a query
BGP	basic graph pattern
$c(n)$	the cost of a triple pattern node
D	a set of data sources
d	a data source
D'	the set of selected data sources of a spanning plan-tree
E	the set of edges in G
e	an edge between two triple pattern nodes in G
E'	the set of edges in a spanning plan-tree
$EstimatedAmount(tp)$	the array that tracks the amounts of matched terms to all variables in a triple pattern
$f(X, L, p)$	pricing function model
$G \models Q$	the query graph of Q

$I(d)$	the increase of intermediate results of a spanning plan-tree contributed by a data source
J	the process of a spanning plan-tree spanning a new triple pattern node
N	the set of nodes in G
$pVar$	a set of variables
Q	SPARQL query
tp	triple pattern
v	a variable
W	the weight of a spanning plan-tree
$w(e)$	the weight of an edge

Declaration of Authorship

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as: S.R. Gunn. [Pdflatex instructions](#), 2001
C. J. Lovell. Updated templates, 2011
S.R. Gunn and C. J. Lovell. Updated templates reference 2, 2011

Signed:.....

Date:.....

Acknowledgements

This has been a long and challenging journey and I would not have reached this far without the help of many important people in my life.

Beginning with my supervisors, Prof. Elena Simperl, Prof. Adriane Chapman and Dr Luis-Daniel Ibáñez, thanks for giving me the opportunity to have this studying experience. I am grateful for your continual encouragement, guidance and expertise throughout the undertaking of my research. I also would like to thank Dr George Konstantinidis for his kind support.

I am also thankful to my colleagues for their kindness and great support in helping me fit in the lab and daily life here in Southampton. I would also like to thank Megan Chan, Belfrit Victor Batlajery, Eddy Maddalena, Amber Bu, Xin Wang, Jacqui Ayling, Lynn Oloro, Tom Blount and Yu Cao. I would also like to mention my appreciation of the help provided by the graduate office, iSolutions and the finance department.

I am extremely thankful to my family, who have been behind me every step of my life: my mother, father, brother, uncle, aunts and cousins.

Chapter 1

Introduction

Since the advent of the World Wide Web (WWW), data has become an increasingly valuable asset in many sectors of the economy (Moor et al., 2019). Around 70% of Bloomberg's \$10 billion revenue in 2018 came from its data access service (B. Taylor and Milton, 2019). In the meantime, the cost of data collection and curation can be prohibitive. Grande et al. (2020) has reported that a mid-sized institution spends more than \$60 million on data sourcing from third parties. This has led to the development of data marketplaces which have emerged as a new category of business of intermediaries that accept tasks, search for data on the Web and return the results found for a fee.

Such data trades depend on being able to establish a match between the data requirements and the available data. Existing work offers good solutions in terms of capturing the data requirements in the form of queries and deriving answers to those queries from the available data via query engines in assorted data market designs. Research on pricing queries has also become well established in recent years, as well as the study of time-efficient query processing. On the other hand, despite being considered a critical aspect of data business, very few studies have been addressed the technical problem of meeting the financial considerations of the participants, i.e. budget constraints for those buying data and profit concerns for those selling it. To avoid the misunderstandings caused by inconsistent terminologies in different studies, this work denotes participants in data trade who sell data as sellers and participants who require data as buyers. We use the term 'marketplaces' for any intermediaries, platforms or agents that provide the service of matching buyers' requirements and sellers' data, whereas we use the term 'data market' for the entire business involving sellers, buyers and marketplaces. A detailed explanation of these terms is given in Chapter 2.

It is common that data buyers concern both with the utility of their purchase and their budget, and that data sellers seek to control data prices, thereby managing their revenue. However, the existing data market designs either fail to take account of these concerns at all or let a marketplace to manage them based on the trust of sellers and

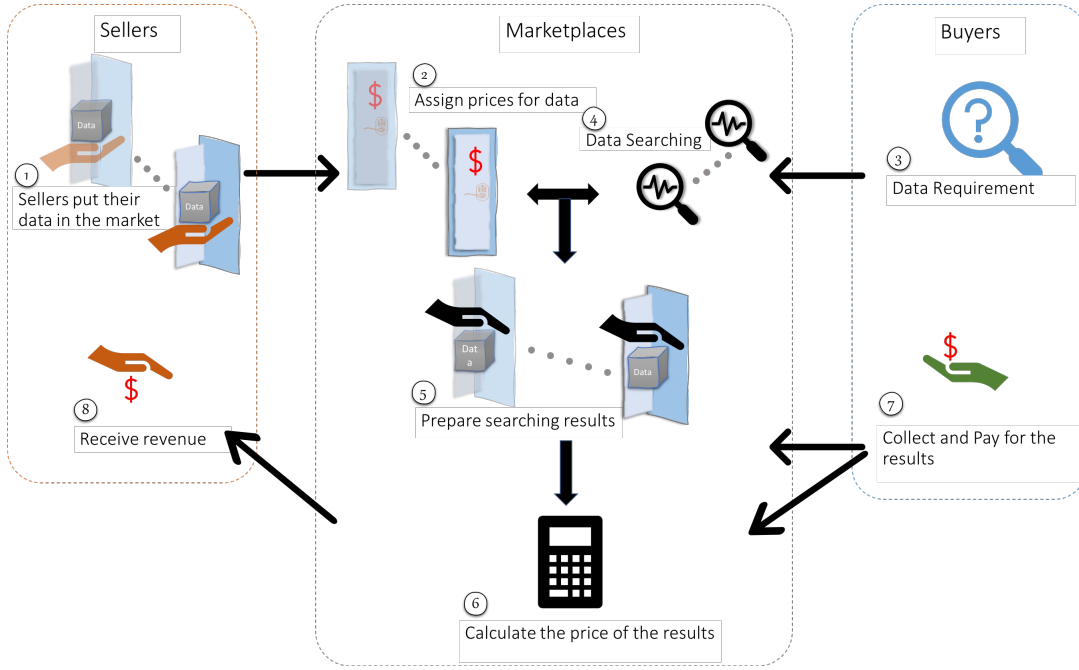


FIGURE 1.1: General Workflow of Data Market

buyers (Li et al., 2017; Grubenmann et al., 2017b). Here, trust refers to (1) fair transactions, in so far as the marketplace does not favour a seller over the other sellers; (2) fair revenue sharing, i.e. that all the sellers get paid in alignment with their contributions; and (3) fair prices, i.e. that buyers get the best results for their money. The extent, or otherwise, of such trust sets limitations: both for the buyers in terms of expressing their willingness to data trade, specifically their budgets and data preferences, and for sellers in terms of controlling their profits, i.e. pricing data and competing against others. Any doubt about this trust will destroy the data marketplace. Instead of fully relying on a marketplace being a trustworthy party, this work considers different methods for sellers to use a marketplace, as shown in Figure 1.2. We explore methods to trade data in a data market and investigate technical solutions to help sellers control their pricing strategies, and buyers purchase query answers under their constraints. This chapter presents our research questions, challenges and contributions. It also outlines the structure of this thesis.

1.1 Research Problem

In recent studies, trading data in a data market involves a sequential process, as shown in Figure 1.1. A data trade starts after sellers place their data in a marketplace, at which point the marketplace assigns prices to the data (Steps 1-2). When buyers bring a data searching task to the marketplace (Step 3), the marketplace will search (Step 4) and prepare the results (Step 5). The price of the results will be calculated (Step 6). Buyers

will pay the marketplace to collect the results (Step 7), and the revenue will be split among the sellers (Step 8).

This design has drawbacks for both sellers and buyers. First, the price of data is controlled by marketplaces instead of sellers. The existing marketplaces either set up specific windows of data, which are called views or data points, for sellers to assign prices or, frequently, they will assign the same price to all the registered data. The specific views and points defined by marketplaces, however, are not flexible enough to capture sellers' perceptions of the differences in the quality, value and collecting costs of the data. Only limited research has been conducted on enabling sellers to price data, and it has to be recognised that if data were to be subject to more varied pricing strategies, this would itself pose a new challenge for buyers. Specifically, with massive data sources in a marketplace with varied prices, buyers face the challenge of choosing where to spend their money, especially when they are subject to budget constraints. However, existing data market designs are not buyer-friendly in terms of the above challenges, which focus only on approaches for pricing or answering queries rather than on buyers' budget constraints. In addition, pricing queries tends to be based on the information revealed by the answers to queries, while ensuring that a pricing strategy is arbitrage-free, i.e. that a query cannot be decomposed into a sequence of queries that return the same answer at a lower price than the original query. Although this is understandable in terms of protecting sellers' profits, an arbitrage-free query pricing strategy is not fair for buyers because it uses the assumption that buyers will conduct arbitrary moves in order to collect data at a cheaper price, which encourages the marketplace to set higher prices to protect sellers. In such a circumstance, buyers who do not cheat are charged for more data than they actually use and are thus taken advantage of.

Furthermore, the above market design requires sellers to trust marketplaces. This means sellers must grant full data access and prices to the marketplace and agree to receive their share only after the marketplace finishes negotiations with buyers. However, such trust is hard to achieve and maintain when the business needs a unit of data pieces from different sellers. For instance, if one seller lost trust in a marketplace for sharing revenue and felt that their data piece was more essential in the business and therefore, deserving of higher revenue, what would happen to other sellers' profits and buyers? Or, in the worst case, if the seller threatened to quit, this would risk destroying the business in the market. Furthermore, can buyers chase the remaining data sources and buy the pieces they need?

While the literature offers solutions in trading data for the circumstances where sellers trust marketplaces, but it does not address situation where there is a lack of trust, as shown in Figure 1.2. For instance, when sellers are willing to put their full trust in a marketplace, they can turn to QIRANA (Deep and Koutris, 2017b) for their data business, whereas sellers that only trust marketplaces for the transfer of payment and revenue can choose FedMark (Grubenmann et al., 2018a). If sellers have no trust in

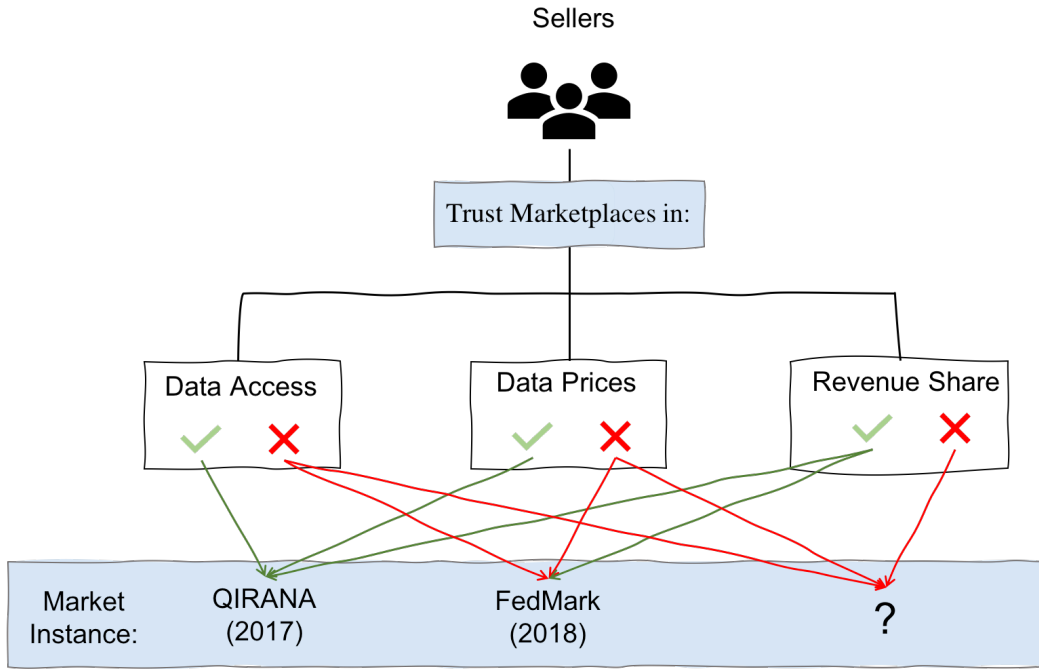


FIGURE 1.2: Trust Options of Sellers

marketplaces, however, their data market options are scarce. Moreover, little attention has been paid to the circumstances surrounding data trading with independent sellers or competing marketplaces together, especially when they apply different pricing functions.

In light of the above-mentioned problems, this work argues that the main obstacles in data trading currently are the unsatisfied expectations of both parties: of buyers to purchase data within their budget constraints, and of sellers to put their full trust in exposing data to marketplaces where they have limited rights in managing data prices and the share of revenue. While there is no denying that reliable giant marketplaces are worth the trust of sellers and are good options for data trade, such marketplaces restrict sellers' control of their data business. For instance, the data pricing strategies, decisions as to whether to trade data, and revenue sharing are out of the control of sellers in the existing markets. In addition, it remains challenging to establish a commonly-agreed query pricing strategy over large numbers of individual sellers. Moreover, while extensive research has investigated affordable and flexible methods for trading data in the form of queries, the answers to such queries that span across massive data sources may still exceed the budgets of buyers. However, if a data market opens the selection of sources and query answers to buyers, the associated introduction of varied pricing strategies of the massive data sources in a marketplace poses a new challenge for buyers in terms of choosing sources to invest their money, especially within a budget constraint. In addition, up to now, far too little research attention has been paid to data buyers who look for data that maximises their utility for a budget, especially when purchasing data over individual sellers and competing marketplaces. Therefore, to resolve

this problem, this work investigates how to improve data trading by enabling diverse data pricing strategies in a data market and assisting buyers to purchase data within their budgets.

1.2 Research Challenges

To narrow the gap occasioned by the differing expectations of sellers and buyers and to solve the above research problem, we propose updating the existing markets with new services to trade data while satisfying both buyers' and sellers' expectations as well as designing a holistic market suitable for the circumstances where there is a lack of trust among individual sellers and competing marketplaces. This section identifies our research questions and the challenges in seeking solutions.

Research Question 1: How to allow buyers to purchase data under their constraints in the existing data markets while sellers can price their data independently?

Research Question 2: How to allow buyers to purchase data from multiple independent sellers and competing marketplaces under their purchase constraints?

Research Question 3: What are the differences between the different types of data markets in terms of their assistance for sellers and buyers?

First, to meet the expectations that buyers can purchase data under their budget constraints, and that sellers can control the data access and prices, a two-step change to the current market are required: opening more pricing options to sellers, and changing their perspective to take account of buyers' legitimate desire to find query answers that maximise their utility for a fixed budget. Furthermore, where there is a lack of trust among independent sellers and competing marketplaces, diminishing the monopoly power of marketplaces is inevitable. We tend to hand the process of assigning price and data searching that is controlled by a marketplace to sellers while collecting the results for the buyers, and while facing the following challenges.

Challenge 1: What pricing options can sellers apply? Although extensive research has been conducted in the field of charging buyers (Deep and Koutris, 2017b; Koutris et al., 2015; Grubenmann et al., 2018b; Moor et al., 2019; Koutris et al., 2013; Grubenmann et al., 2018a), how to price data or queries remains an open question. The established ideas in the previous markets are either to define a basic unit of data (such as a view and point) and assign a fixed price to these individual units and then calculate the price of a query based on the units accessed or required, or let the participants negotiate. Several attempts have been made to refine the calculation strategy and the negotiation method in order to benefit sellers or buyers, but finding a pricing method that is well accepted and suits various circumstances is still an open problem.

Challenge 2: Manage data duplication among multiple data sources assigned with a wide range of prices. In the absence of the prices of duplicated data, it has been straightforward for a marketplace to select one or multiple sources to collect the duplicated data and generate query results. However, varied price settings for sources with overlapping data makes the selection challenging (Dustdar et al., 2012). Concerning the method of charging query answers, collecting duplicated data from multiple sources indicates overpaying for the same data, which is not fair for buyers. Although avoiding sending sub-queries to unnecessary data sources to collect duplicated data can overcome the overpaid issue and improve query plans, designating a unique source for duplicated data is complicated in the presence of varied prices, especially when the selection has a budget constraint. Very little work has touched on this challenge and such as it has been limited to simple pricing methods.

Assumption 1: Open information about data sources before access. When marketplaces have insufficient trusted access to data sources to act as an intermediary, buyers face the challenge of solitary searching foresight with zero knowledge about massive data sources in a marketplace with varied prices. This could force them to search all the available sources at a great cost in time and money. Given that Grubenmann et al. (2017a) claim the impossibility of selecting sources with zero information for searching tasks, in this work we assume that there is a public way to collect basic information before trading data, e.g. metadata or statistical knowledge.

Challenge 3: Estimate the prices of joining results. Since sellers can charge for data access independently by applying varied pricing strategies, it is essential to estimate the price of each sub-query sent out by the executor before query planning, especially when buyers are aiming at cost-efficient query planning. However, join estimation has been a challenging problem since the beginning of query languages and has no perfect solution as yet. No known research on estimation has considered the prices of data specifically. The majority of studies try to minimise the calculation cost. When data have varied prices, however, although joining the intermediate results of two sub-queries can result in the same calculation complexity, the price of the intermediate results could differ if a buyer purchases the intermediate results of a sub-query first, and then uses a bind join to search for the results of the other sub-query.

1.3 Contributions

The aim of the research in the present thesis is to fill in some of the gaps in the current data markets so as to meet the expectations of sellers and buyers for trading data. To this end, this work provides an overview of the existing research in the field of data markets, highlights the trust issues among participants, presents a new insight into the field and contributes methods to maximise the utility of buyers for a budget. The performance of all the work is examined by the conducted experiments.

We adjust the workflow of the current data markets into a federated marketplace, Data Emporium, to satisfy the requirements of both sellers and buyers. Data Emporium widens the accepted strategies for data prices and enables buyers to purchase the required data according to their budgets by

- (1) analysing the impact of pricing functions in data purchases;
- (2) discovering the new challenge of a specific type of pricing functions - access dependent pricing functions (ADPFs);
- (3) modelling the problem into two sub-questions: cost compression and allocation;
- (4) demonstrating heuristic algorithms to approximate the optimal solutions.

To address the non-trust issues among the participants in a data market, we propose an architecture, *Free Market*, where data is traded between sellers and buyers with a central marketplace that is less involved. The workflow of data trade in the Free Market will allow sellers to price their data autonomously; let the buyers purchase data directly from wanted sellers; and assist buyers to purchase lower-priced answers to their queries through a local query engine. In that regard, since buyers do not have a trust party to rely on to make a data purchase on their behalf, to reach the goal of assisting buyers to purchase lower-price answers, we design and implement a local query engine on the local side of buyers which engages buyers to set up preferred data sources and then accesses the public scheme/metadata and prices of the data sources before accepting queries with budget constraints. Intending to serve buyers' expectations of lower-priced query answers within their budget constraints, this work enables the local engine to:

- (5) model queries and their plans into graphs and spanning trees;
- (6) estimate the prices of data as the weights of edges in graphs and trees;
- (7) generate the price-based plans of queries by searching minimum spanning trees to lower the cost;
- (8) test the efficiency of the price-based query planning using a widely-accepted benchmark.

1.4 Thesis Structure

The remaining part of this thesis comprises six chapters. Chapter 2 presents an overview of studies in the field of data markets based on our proposed architecture of a data market in which the abstract basic components are considered in theory and practice, to date and in the near future. In the meantime, we discuss the organisational structures of

data market implementations and describe the data sources and services, data requirements, data searches and data deals within a data market and summarise the instantiation of each component in the current work in Chapter 2, along with discussions about the unsatisfactory expectations of buyers and sellers in previously-published studies.

Chapter 3 is concerned with the methodology applied in resolving the research questions of this work. It describes our plans, first to explore a method to allow buyers to purchase data within their purchase constraints in the existing data markets while simultaneously allowing sellers to price their data autonomously, and then, second, to design a data market where buyers can purchase data within their constraints from multiple independent sellers (i.e. individual sellers or marketplaces), and, third, to compare our work with existing studies with respect to buyers' and sellers' satisfaction with data market services. Following the research plan, the chapter introduces the data structure and basic algorithms applied in our research and the design of the evaluations of our work.

Chapter 4 demonstrates the updated data market, Data Emporium, with our heuristic approaches to satisfy the expectations of sellers and buyers. The chapter first introduces Data Emporium, which allows sellers to apply different types of pricing functions. Then, it generalises the problem of purchase allocation with budget constraints, analyses its complexity of the problem and specifies the new challenge inherent in this in respect to ADPFs. After that, it introduces our two-step approach to solving the problem: a cost compression algorithm to minimise the cost of intermediate allocations, and Greedy and 3DDP heuristics to approximate the optimal allocation. An experimental evaluation of the performance is then presented, followed by a summary of the work presented in the chapter.

Chapter 5 lays out the design of Free Market, and its a local query engine. In this chapter, Free Market, as a free business environment, is provided for all kinds of potential sellers and buyers. Sellers reserve the management of their data sources in terms of access control and pricing functions, while the marketplace in Free Market only makes an announcement of the available data sources. Buyers independently search for answers to their requirements and directly trade data with sellers via a local query engine. The concept and design requirements of the local query engine is explained in the context of a buyer's data requirement, demonstrating how it works in a different way from the existing query engines so as to address the challenge of improving query plans to meet the constraints of budgets and preferences/priorities.

Chapter 6 points out the problem of assisting buyers to purchase lower-cost query answers in a free market and then proceeds with our approach for the implementation of a local query engine. The problem is how to generate a query plan based on the prices of data to reduce the cost of query answers. To resolve the problem, we model it as a

minimum spanning tree problem with the cost of data requests as the weight of spanning trees, propose heuristic approaches to approximate the optimal query plan for a local query engine and evaluate their performance in comparison to the state-of-the-art query planning algorithm.

Following the work in Chapter 6, Chapter 7 addresses the issue of assisting buyers to purchase data under budget constraints in Free Market. It first describes the problem and the formalisation of it based on the approaches in Chapter 6, and then demonstrates our approximate algorithms for solving the problem, with the discussion about their efficiency based on the conducted experiments.

Chapter 8 examines the different structures of data markets and focuses the discussion on their services for sellers and buyers from the following aspects: (1) Management, (2) Trust, (3) Pricing, (4) Dependency, (5) Budget, (6) Efficiency, (7) Searching Process, (8) Settlement.

Finally, Chapter 9 concludes this work, discusses its limitations and suggests future directions of following research.

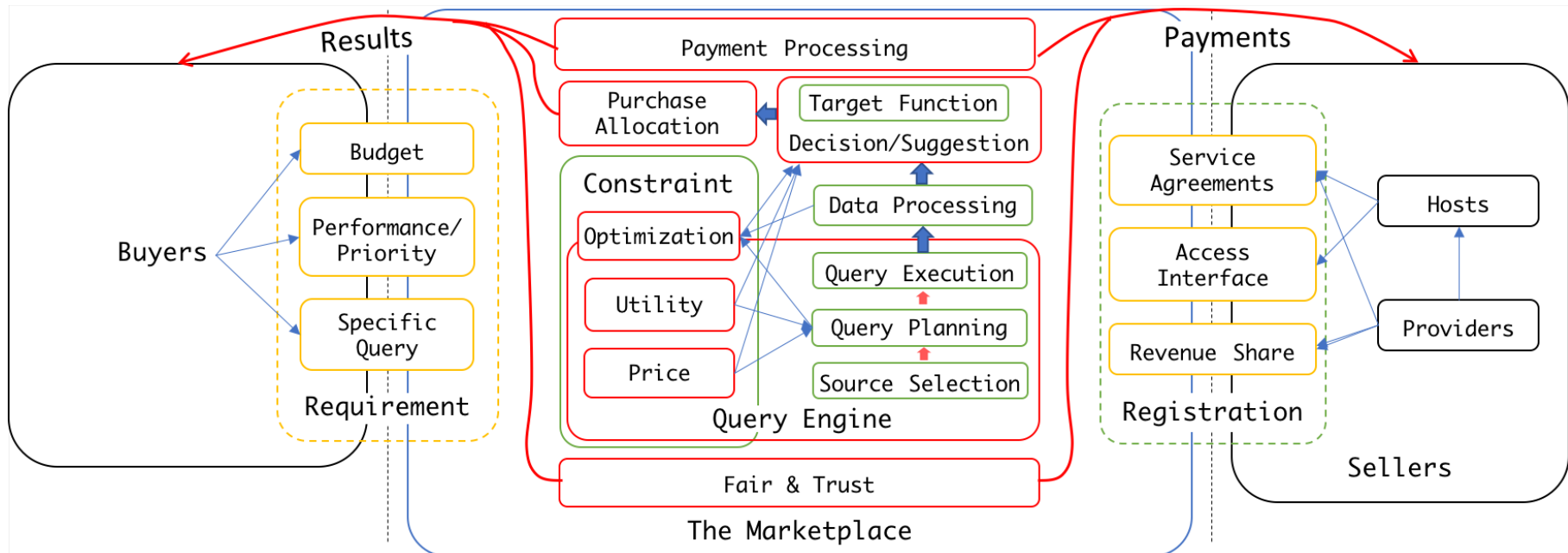
Chapter 2

Literature Review

Both industry and academic communities have called for a market for information goods like data for decades. [Barrett and Konsynski \(1982\)](#) first introduced an electric information sharing system which was later called *information marketplace* ([Armstrong and Durfee, 1998](#)). [Wang and Archer \(2007\)](#), explicitly defined the system as an electric market (EM) and reviewed the EM literature. With the fast development of the Web, a new category of business of information intermediaries emerged that accept tasks, search the Web and return the results found for a fee ([Florian et al., 2014](#)). To the best of our knowledge, in 2008, [Wu et al. \(2008\)](#) were the first to use the term *data marketplace* to refer to a system that browses and selects disparate data sources over a digital data storage network. Since then there have been numerous papers designing data marketplaces for various scenarios ([Ardestani et al., 2015](#); [Green et al., 2007b](#); [Dunning and Kresman, 2012](#); [De Vos et al., 2019](#); [rae, 2018](#)).

This work discusses the gap evident in current data market models between the requirements of sellers and buyers in the current data market models in an effort to identify ways to facilitate data trading. To this end, this chapter presents an architecture of a data market so as to outline the basic components and the relationships among them in the existing data markets. This is then used to explain the participants in data markets and the structure of data markets, and then this chapter reviews the instantiation of those data market components in previous work and identifies the gaps in the current knowledge before concluding with a summary.

FIGURE 2.1: Architecture of Data Markets. Yellow components are the information accepted by a data marketplace. Red components and arrows are the process of trading data in a marketplace. Green components imply that the instantiation requires optimisation.



2.1 Architecture of Data Market

Figure 2.1 illustrates the architecture of a data market. The rectangles represent the components of the data market, whereas the arrows showcase the relationships among the components and the workflow of trading data in the market. Note that the components are either abstractions of functions in the designs and models of the existing data markets or hypothetical requirements in the future. This architecture is the basis for this literature review although not all the components illustrated are considered in the literature discussed in the following.

2.1.1 Participants in Data Trade

Three roles interact in the data trade within a data market: data buyers, data sellers and marketplaces. Both buyers and sellers are the clients of marketplaces who are looking for data business opportunities. Marketplaces act as intermediaries for their clients.

- **Data Buyers:** These are the participants in a data market who have data requirements and are willing to purchase data. Their requirements consist of a specific query that explains the desired data, preferences/priorities in respect to available data and a budget that sets the limit for the cost of a data purchase.
- **Data Sellers:** These are the participants in a data market who have data and intend to join in data business and provide data and/or data services for revenue. They register their data in a marketplace and reach the agreements for the provision of services, including but not limited to an interface for data/services access and a clear method for calculating the revenue share of sellers.
- **Data Marketplace:** This is a party or a platform that addresses the requirements and registrations of data buyers and sellers and operates transactions for data purchases. It integrates a query engine, trust regarding fair behaviour, components to satisfy the constraints of data requirements, and processes payments.

In this work, we assume that all the roles present their desires in the most effective way. Data buyers and sellers can express their expectations in the form of queries, approaches to ranking and valuing data as well as methods of data storage and access for different types of data, such as investigation data, self-generated data, non-customised crawled data or raw data (Stahl et al., 2014). Marketplaces have the expertise in processing queries and payment transactions.

Related surveys and classifications of data markets use other terminologies for the participants, but still in essence follow the similar descriptions to ours. For example, Zuidervijk et al. (2014) presented a similar description for the *data buyers'* role but used the term *data users* instead. In some cases, data users and the parties that pay for data are different, such as sponsors of free data or services. In this work, we use

the term, *buyers*, to represent both. Liang and Huang (1998) and Florian et al. (2014) referred to data buyers and sellers as customers and vendors, respectively. Using the term, Data-as-a-Service (DaaS) (Truong and Dustdar, 2009), Grubenmann et al. (2017b) considered data owners and DaaS providers as separate participants and defined them as two separate roles on the seller's side: *Provider* and *Host*. Under their definition, *providers* are claimed to be the data owners who register data on a data marketplace and intend to sell it, while *hosts* are the ones hired by *providers* to provide query execution, data delivery and other agreed services. Dustdar et al. (2012) refer to data owners as *data publishers*. Marketplaces, meanwhile, have been referred to, variously, as infrastructure, frameworks and platforms in the work of Stahl et al. (2016), Florian et al. (2014) and Moor et al. (2019). Deep and Koutris (2017b) and (Fernandez et al., 2020) named the role of marketplaces as data brokers and arbiters, respectively. Differently, Li et al. (2017) define the participants who negotiate between buyers and sellers as market makers.

The foundation of the business relationship among the three participants is trust. Following the same concept of trust in digital ecosystems as defined in Ozer and Zheng (2017) and Cioroica et al. (2019), as users of a service provided by a marketplace, buyers and sellers are trustors, making the marketplace a trustee. Marketplaces should not treat sellers and buyers unfairly when they make 'self-serving' decisions that stem the profit increase of sellers and buyers. Furthermore, sellers and buyers should be able to trust a marketplace to satisfy their requirements and expectations according to a trustum Ozer and Zheng (2017) while using the services. Recent studies have used this relationship as an assumption in the designs of a data market.

2.1.2 Structure of Data Market

In terms of data management and operation, the existing data markets have two organisational structures: centralised and federated.

- **Centralised Data Market:** This kind of market relies on one individual marketplace to manage data, process queries and make decisions on data and its prices when trading between buyers and sellers. In other words, sellers and buyers only participate in the data business by providing data and raising requirements, respectively, whereas the marketplace take on the entire process of the data trading.
- **Federated Data Market:** This kind of market enhances the involvement of sellers in the data business by dividing the responsibility for the data management and operations between sellers and marketplaces. While a marketplace is still in the position of collecting data for buyers, sellers can choose to manage their data and operate query execution via collaboration with the marketplace in a federated market.

In a centralised data market, all sellers locate data in a marketplace and authorise it with the right to access and sell data. When a requirement from a buyer triggers the trade of data, the marketplace processes the computation of query results for buyers' data requirements, prices the results and exchanges them for payments and then distributes the payments among the involved sellers. This structure is capable of trading data effectively, since the central marketplace hosts all the data and has the necessary information to optimise query operation. On the other hand, it is arguable that the marketplace in a centralised structure operates as a monopoly. The drawbacks are: (1) that buyers can be overcharged and suffer disadvantages due to the fact that marketplaces' strategies for pricing query results are set to maximise revenue for sellers or for the marketplaces (Chawla et al., 2019b); (2) that sellers are not able to decide autonomously the price for selling their data or to monetise their data when the data is duplicated with that of other sellers'.

Alternatively, a federated data market embraces sellers who independently host data, operate query executions and price the query answers. Given a data requirement, the marketplace in a federated data market chooses sellers to participate in the computation of a query answer and sends out decomposed queries to the sellers, before waiting to collect the results that are returned, i.e. intermediate results, with the asked prices. These intermediate results are then used to generate the answer to the original query and price it as the sum of the asked prices. The federated structure offers flexibility to sellers in terms of data management, query execution and data prices. On the other hand, if data trading requires prescient knowledge about data from the federated sellers in order to choose among them, the federation of sellers can mean that it takes longer to generate answers to queries due to the extra processing time.

The following sections discuss each component in turn so as to describe the instantiation of the above structures in the existing data markets.

2.2 Data Sources and Services in Data Market

Data sources and services are essential in data trading as well as the concerns about monetising data (Truong and Dustdar, 2009). The data trade begins with the collection and maintenance of data. Despite extensive research, the methods for data collection and operations for the management of data remain expensive in terms of both effort and money. Data services addressing the issue of an appropriately restricted publishing method of data sources for buyers to access and use (Carey et al., 2012). In the data business, services are also built up at a fee, in addition to the cost of data itself. Hence, the settlement for the following components is a prerequisite for sellers to explicitly address when they bring their data sources into a data market:

- **Service Agreements:** These settle the agreed arrangements for the data services that sellers required in a marketplace. The reason for having a service agreement is to stipulate the method for data sale and protect the benefits accruing to sellers. The service agreement answers the question of how to sell data and sets rules for the prevention of data leakage, loss, tamper or arbitrage when data sources are exposed in a data market. [Aljamal et al. \(2018\)](#) provided a comparative review of service infrastructures. In this work, we follow the general practice in the literature of assessing service agreements according to four aspects: storage, functions, performances and payments.
- **Access Interface:** This is the method for accessing data that either sellers or the marketplace supports. The three widely-adopted interfaces are (1) SPARQL Endpoints, which a SPARQL protocol service is listening at ([Feigenbaum et al., 2013](#)) and are capable of processing queries and returning answers; (2) an Application Programming Interface (API), which enables programs to communicate with a remote operating system or control programs (e.g. DBMS); (3) Download File (DF), provides download links. [Stahl et al. \(2017\)](#) showed that 18.1% of the providers offer only one access type, 41.7% two, 27.8% three, while only 8.3% offer all access types.
- **Revenue & Share:** This defines two factors—objects and pricing—to explain the method by which sellers or a marketplace charge buyers for using data, and distribute revenue. This is a fundamental statement for the data business since it serves to clarify the model to gain profit. Pricing interprets the strategy for calculating the cost of data requirements and the agreed approach for sharing profits if multiple sellers are involved in the trade. The objects are the entities that pay the cost.

Table 2.1 classifies and summarises work related to the settings of the above-mentioned components. The table sets out the design of each study and how it fulfils the *storage*, *function*, *performance* and *payment* elements in service agreements, as well as *access interface* and the *objects* and *pricing* of revenue and share. Specifically, *storage* identifies the data that a marketplace stores or operates, which can be either the complete data sets, metadata or some structured summary statistics of the data sources; *function* is the service offered to buyers, for instance, query, data integration, data updates, etc.; the *performance* column lists the prioritised goal that the data or services are aimed at, such as the responding time and answer completeness; *payment* denotes the agreed fee for marketplaces serving sellers which are marked as 'Y' in the table if the cited study considered it; otherwise, it is empty (The discussion of charging for services is out of the scope of this work. Interested readers please refer to the work of [Thanakornworakij et al. \(2012\)](#); [Wang et al. \(2016\)](#)); *access interface* defines the method to access data sources; *objects* are the entities targeted as source of revenue, which can be the direct users of data sources or sponsors; and *pricing* indicates the strategies that sellers

or marketplaces adopt to achieve profits. Regarding the scope of this work, we will focus on explaining the meaning of the cell details for the component of revenue & share. Note that blank cells in Table 2.1 imply that the corresponding work has not considered that specific component.

TABLE 2.1: Literature Summary on the Research of Data Source

Research Work	Data Sources						
	Service Agreements				Access Interface	Revenue & Share	
	Storage	Function	Performance	Payment		Object	Pricing
Grubenmann et al. (2018b)		Search	Latency		SPARQL Endpoint	Sponsors	Delayed-answer Auction
Deep and Koutris (2017b)	Data Sources	Query	Completeness			Buyers	Query-based Pricing
Grubenmann et al. (2017b)	Data Source	Query	Data Quality	Y	SPARQL Endpoint	Buyers	
Li et al. (2017)	Data Source	Query			SQL Endpoint	Buyers	Balanced Pricing
Roman et al. (2017)	Data Source	Enrichment	Advertise	Y	API	Buyers	
Stahl and Vossen (2015)							Quality Discount
Zuiderwijk et al. (2014)	Data Source	Quality Assessment Data Processing Tools Multilanguality etc.			API	Buyers, Funders	Donate, Free
Balazinska et al. (2013)						Buyers	Automatically Derive based on pricing schemes
Dustdar et al. (2012)	Metadata	Integration	Reliability, Availability, Security		Service Interface	Buyers	Quality-dependent Pricing

2.2.1 Pricing Query Answers and Services

Since data trading is a relatively new business, it is vital to establish a model for pricing data and services, including for revenue and share. In terms of pricing query answers or the service of searching data, the existing research in data markets has adopted the different forms in network economics (Shapiro et al., 1998). We classify these studies into two categories based on the approach they used to decide the prices: negotiated pricing and multi-factor pricing. We then explain and discuss the respective instantiations in the current work as follows.

In this thesis, we clarify the prices, revenue and share in data trading: *pricing strategy* is the mechanism that sellers or a marketplace choose to raise revenue and to calculate the share of each seller; the specific method for calculating the price of data or query answers is referred to as a pricing function, which takes the required data as its input and returns a non-negative number as the price of the data.

2.2.1.1 Negotiated Pricing Strategy

Negotiated pricing strategy implies that the prices of data or services are decided by negotiations between sellers or marketplaces and buyers. The basic idea behind this is that the value for data and services is subjective to its buyers (Shapiro et al., 1998) and, thereby, the pricing strategy should allow for a disclosure of the buyers' true willingness to pay (Stahl and Vossen, 2015).

Name Your Own Price is an approach for a negotiated pricing strategy adopted in an auction. Stahl and Vossen (2016) implemented a model of this approach that allows sellers or a marketplace to name an initial price of their data, but if a buyer refuses to pay that price, then the buyer can name another price that she or he is willing to pay. Next, the marketplace computes the reduction in both the size and quality of the answer to meet the new price. The problem is formalised as a Multiple-Choice Knapsack Problem, and a Greedy algorithm is proposed as a solution. (Grubenmann et al., 2018b) proposed to let sponsors promote specific data by bidding a smaller delay in answering queries than data with lower bids.

2.2.1.2 Multi-factor Functional Pricing Strategy

A multi-factor functional pricing strategy is a strategy that use a well-designed function to set the price of the required data or services and sellers' shares. The existing multi-factor pricing strategies follow a common four-step design: *basics*, *desiderata*, *formula* and *share*. Generally, a multi-factor pricing strategy takes a query or a series of queries as input, according to the definition of *basics*, applies varied processes to satisfy

a trade-off among the desiderata and then calculates the price of the input by a formula before sharing the received payment among the sellers when there are multiple sellers involved in the transaction.

Basics *Basics* are the basic definition in the design of multi-factor pricing strategy. It clarifies the basic unit of data and the contributing unit for answering queries.

- **Data Unit:** This is the basic item to measure the size or amount of information. The widely applied basic units of relational data are versions, views and points (Balazinska et al., 2013; Deep and Koutris, 2017b), whereas the resource description framework (RDF) (Cyganiak et al., 2014) data uses a *triple* as its basic unit.
- **Contributing Unit:** This refers to a set of data units that contribute information to answer queries. A straightforward definition is that the set of data units that have been accessed while answering a query is the contributing unit of the query (Upadhyaya et al., 2016). Moreover, Deep and Koutris (2017b) captured the contributed information by the dependency of prices on queries, query outputs and the data support set, while Tang et al. (2013) defined it as the set of minimal provenances of query answers.

Desiderata *Desiderata* is a term referring to several commonly-agreed desirable features to consider from the perspective of both sellers and buyers for a design of a multi-factor pricing strategy (Grubenmann et al., 2018a; Deep and Koutris, 2017b; Koutris et al., 2013; Lin and Kifer, 2014; Deep and Koutris, 2017a; Chawla et al., 2019a; Lin and Kifer, 2014).

- **Non-overpaid:** Buyers should not be asked to pay for the same information more than once in one data purchase.
- **History-awareness:** In the case of requiring answers for a series of queries, the price of the following answers should be aware of the paid information and not charge for it again.
- **Arbitrage-free:** The price of a query answer cannot be greater than the sum of the prices of a series of queries' answers when the contributing unit of the query is a subset of the union of the series queries' contributing units. This aims to protect the profits of sellers.
- **Customisability:** Sellers can customise the pricing functions as much as they want while buyers are able to purchase only the data they need instead of being forced to purchase entire data sets.

The trade-offs among these desiderata shape a wide space for the design of multi-factor pricing strategies.

Formula *Formula* is the process that satisfies a trade-off among the desiderata and the methodology that conducts the calculation of the prices of input queries. There are two types of formulas in the existing work: data-based and query-based.

- **Data-based Formula:** This first defines a function to price the basic data unit, and then calculates the price of a query by summing up the prices of the data units in the contributing unit of a query (Grubenmann et al., 2018a). Stahl et al. (2014) classifies the method of pricing the basic unit into four categories: free, freemium, price-per-unit and flat rate. The *free* category simply is simply the method that opens the data access for free; *Freemium* allows free access to data units up to a certain threshold but charges a flat fee if more data units are required; *Price-per-unit* assigns a fixed value per data unit as its price; *Flat rate* charges for all data access at a fixed price (Muschalle et al., 2012; Schomm et al., 2013; Liang et al., 2018; Mehta et al., 2019; Muschalle et al., 2012).
- **Query-based Formula:** Koutris et al. (2015) defined the query-based pricing formula as an approach that first allows sellers to set explicit prices on individual data units, or sets of data units, before automatically deriving the price of query answers based on the explicitly priced data units and the feature of queries. In contrast to data-based formulas, a query-based formula is designed to distinguish the price of queries when the numbers of accessed data units or query answers are the same but those queries are different, especially the aggregation queries. This model leads to numerous advance research in theory and practice (Koutris et al., 2013; Deep and Koutris, 2017a; Lin and Kifer, 2014; Deep and Koutris, 2017b; Chawla et al., 2019a).

Tang et al. (2013), Koutris et al. (2015) and Li and Miklau (2012) analysed the complexity and difficulty of computing and conforming a query-based pricing formula that satisfies the desiderata which is at NP-complete in varied cases and provides heuristic solutions. Deep and Koutris (2017b) designed four functions with two query-based formulas to protect sellers from arbitrage while enabling history-awareness. Upadhyaya et al. (2016) proposed lightweight modifications to data APIs to achieve an optimal history-aware pricing strategy so that clients are charged only once for data that they have purchased and that has not changed. Their idea is to allow buyers to ask for refunds for the data they have purchased before.

Regarding the satisfaction of the desiderata, Deep and Koutris (2017b) claims that data-based formula functions that assign prices to query answers based on the size of outputs, or on the provenance, offer no protection against arbitrage. However, when the calculation of a contributing unit is based on the original data instead of the returned query answers, the data-based formula can be arbitrage-free. In addition, Grubenmann et al. (2018a) proposed a data-based formula that avoids buyers paying more than once for duplicated data in a contributing unit.

It is arguable that both formulas protect the benefits of both sellers and buyers, especially sellers when a formula is arbitrage-free. That said, in a marketplace where sellers offer duplicated data units with different prices, arbitrage-free will be unfriendly and unfair for buyers searching for a cheaper answer, especially when they have limited budgets. Furthermore, buyers still have no choice but to pay for the entire set of query answers except in the case of the formula in FedMark (Grubenmann et al., 2018a). In addition, the refund policy (Upadhyaya et al., 2016) even adds extra work for buyers since they have to ask for refunds proactively. Moreover, the complex query-based formula makes it difficult to offer transparency to buyers.

Share Distributing payments for the sold data among sellers is the last step of a multi-factor pricing strategy. Considering the scenario when the contributing unit of a query consists of data units from multiple sellers, it is necessary to share the payment fairly between these sellers. Although distributing the profit to sellers can be simply paying each seller the sum of the prices of the contributing unit that she or he owns. However, that is only the case when no duplicated contributing data exists in the data-based formula. In addition, it is hard to calculate the prices of contributing units of individual sellers in query-based formulas.

Koutris et al. (2013) presented a solution, *FAIRSHARE*, which shares the revenue among multiple sellers when the contributing data is not unique, and avoids sellers gaining no profit when they have the same contributing data but are not sold to buyers. In this system, however, sellers have no choice but to agree with the sharing policy, indicating that they always share revenue with others who own the same data regardless of the costs they have incurred in data collection and maintenance.

2.3 Buyers' Requirements for Data

Data buyers require data from a data marketplace and intend to purchase data that satisfies their requirements. There are three elements governing a precise expression of buyers' requirements:

- **Budget:** this is the maximum amount of money available a data purchase. It can be limited to a given value, set to be as little as possible or infinite. We argue that the budget is a crucial factor affecting buyers' purchase decisions, especially when they are not allowed to customise a query answer to meet their budget but pay for the entire data set or any returned answers.
- **Preferences/Priorities:** These express the buyers' appraisal of data from the perspective of their preferred data type, source and quality (Zaveri et al., 2015; Thakkar et al., 2016), or the performance of services, i.e. speed, accuracy and correctness. It is arguable that buyers may not have an accurate way to rank, rate or value

TABLE 2.2: Summary of the Research on Data Requirements

Research Works	Data Requirement		
	Budget	Performances/Priorities	Query Type
Balazinska et al. (2013)		Data Accuracy	SQL
Li et al. (2017)		Data Privacy	Linear Aggregation Query
Grubenmann et al. (2018b)		Answer Latency	SPARQL
Dong et al. (2012)	Y	Coverage & Accuracy	Other
Zollinger et al. (2012)	Y	Data Quality	SPARQL

data that they have not yet seen. However, the growing research interest in the user preferences regarding data and services (Miller and Mork, 2013; Yeganeh et al., 2014; Opresnik and Taisch, 2015) renders this element necessary and predicts that it will be functional in the future. In this work, we cast the buyers' preferences/priorities as the data utility to buyers.

- **Specific Query:** this formally describes the data requirements in a query language, such as SPARQL (Harris et al., 2013), SQL or others. In addition, the requirement can also be formed into a program to access data via APIs. In this work, we assume buyers work their best to form their requirements into a professional query and skip the discussion of transforming questions into queries. Buyers are looking forward to purchasing the answers to the specific query from a data marketplace. This work uses the term 'solution' to refer to a single returned result after processing a query, whereas the 'answer' to a query is a set of distinct solutions.

Table 2.2 shows a summary of works relevant to the above-mentioned elements of data requirements. For each article, the table identifies its design and how it fulfils the *budget*, *performance/priority*, and the type of *queries*. In detail, the notes under the budget column indicate whether the corresponding work has considered *budget* which is marked with 'Y' or not which is left as empty. The preferences/priorities column lists the categories of the specified expectations of buyers, which can be data quality and/or types of services. Query type indicates the query languages that the corresponding work considered. This is left empty when no concrete type is identified in the respective study. Compared to Table 2.1, it is evident that limited work has considered the requirements of buyers or markets that are designed as buyer-friendly. The focus of the existing data markets has been on the structural design and pricing strategy. The expectation of buyers, i.e. to purchase answers within a limited budget, possibly while maximising the criteria of buyers' profits, has not been seriously considered or even ruled out in the work of Deep and Koutris (2017b); Koutris et al. (2013, 2015); Muschalle et al. (2012); Balazinska et al. (2013); Li and Miklau (2012).

2.4 Data Searching in Data Markets

In the current data market designs, marketplaces are in the position to search all data sources for the best match to a data requirement of a buyer. This is because both buyers and sellers trust the marketplaces to arrive at fair matches (Li et al., 2017; Grubenmann et al., 2017b) and to have knowledge about all the data sources which are available for sale. One efficient method to conduct such a search is to adopt a query engine to execute the specific query in line with a buyer's requirement. Furthermore, there can be constraints for the execution due to the specified budgets and preferences/priorities along with the query in the requirement.

- **Query Engine:** This sits on top of the database or the data storage system of a marketplace, thus, its type is associated with the structure of the marketplace. In general, the centralised marketplaces adopt a query engine to execute queries over one database that stores all the data together (Deep and Koutiris, 2017b), whereas federated marketplaces require a federated query engine to support query execution over federated data sellers.
- **Constraints:** these are the limitations for answering queries set by the requirements of buyers. Recall the discussion of data requirements in Section 2.3. The possible constraints are from a limited budget and an expression of the buyer's preferences/priorities. A viable way to embed these constraints into query execution is to formalise them into a price constraint and an optimal goal of utility, respectively.

TABLE 2.3: *Dataset Marketplace* side literature

Current Works	Dataset Marketplace								
	Query Engine			Constraints			Data Processing	Decision /Suggestions	Purchase Allocation
	Source Selection	Query Planning	Query Execution	Performance	Price	Utility			
Grubenmann et al. (2018a)					Y	Y		Allocation Rules	Y
Li et al. (2017)					Y			Balanced Pricing Framework	
Grubenmann et al. (2017a)	Y	Cardinality	Y	Y			Join		
Zuiderwijk et al. (2014)			Y				Visualizing, Comparing		
Dong et al. (2012)	Y				Y	Y	Integration	Marginalism	
Hagedorn and Sattler (2014)									
Zollinger et al. (2012)	Y		Y		Y	Y		Bid&Plan	
Roman et al. (2017)							Transformation Visualisation		
Dustdar et al. (2012)					Y			quality-driven querying	

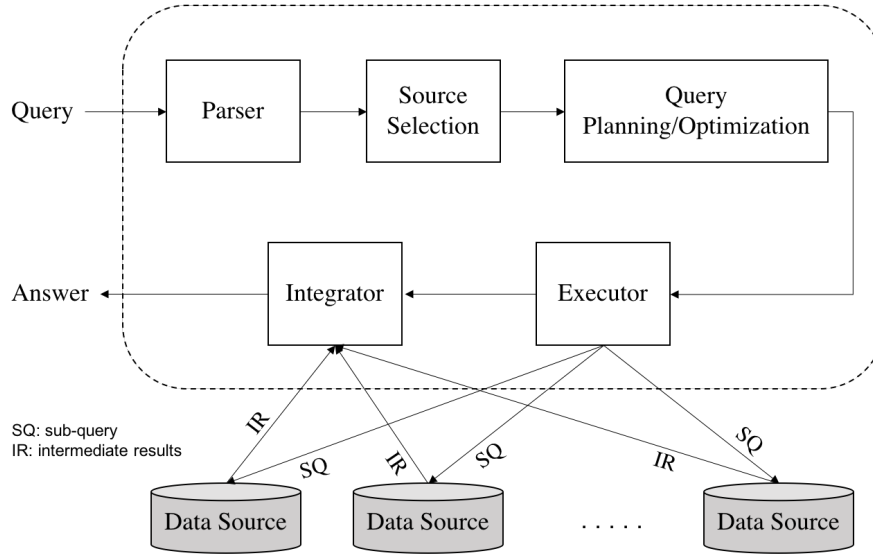


FIGURE 2.2: Processes of A Federated Query Engine Collecting Answer to A Query

Table 2.3 shows a summary of the literature related to the role of *Data Marketplace* in data markets, as illustrated in Figure 2.1. This section discusses the components related to data searching, which are query engine and constraints in Figure 2.1 and the corresponding columns in Table 2.3. For the work listed in Table 2.3, the abbreviation ‘Y’ in a cell under a column implies that the searching progress in the work had considered the corresponding components denoted by the column; otherwise, the cell has no content. Other components of data marketplaces in Figure 2.1 and the corresponding columns in Table 2.3 will be explained in the next section.

2.4.1 Federated Query Engine

Federated query engines have been studied for decades (Lim and Srivastava, 1993; Deshpande and Hellerstein; Schwarte et al., 2011; Görlitz and Staab, 2011; Buil-Aranda et al., 2013; Acosta et al., 2011; Saleem and Ngomo, 2014; Qudus et al., 2019). In this work, we focus on the satisfaction of a federated query engine to its work in a data market. We present a brief review of general steps in federated query processing, followed by a discussion of the existing studies on the optimisation of query execution and their limitations regarding the possible constraints in data requirements.

2.4.1.1 Query Processing

Given a query, federated query processing involves three general steps associated with data sources: source selection, query planning and query execution, and another two steps: parsing and integration, as shown in Figure 2.2.

- **Source selection:** this is the first step to process a query federally. Regardless of the differences in the fields of the federated SPARQL query (Grubenmann et al., 2017a) or relational data integration (Dong et al., 2012), source selection aims at selecting a subset of available data sources in a marketplace that are relevant to the query. This means that the sources include the data that can contribute to answering the query, and can process the query when it is requested (Callan et al., 2017; Powell et al.; Paltoglou et al.; Grubenmann et al., 2017a).
- **Query Planning:** this generates the order in which sub-queries are sent out to be executed in the selected data sources, which is called a query plan (Charalambidis et al., 2015; Saleem et al., 2018). A query plan is both time and price-efficient in order to reduce the running time of execution and the cost of query answers.
- **Query Execution:** this is the process that executes a query according to its query plan (Hartig and Ozsu, 2014). The process involves sending out sub-queries to the selected sources, letting individual sources answer a sub-query, and then collecting intermediate results from the federated sources.
- **Data Integrator:** this is the procedure conducted on intermediate results returned by data sources to produce query answers. With respect to the aggregate queries, it consists of, but is not limited to, data joining, union, ranking and integration while solving data fusions (Michelfeit et al., 2014; Pochampally et al., 2014; Dong et al., 2012; Grubenmann et al., 2018b). Zuiderwijk et al. (2014) and Roman et al. (2017) can transform, compare or visualise data according to enquiries.

2.4.1.2 Query Optimisation

Optimisation of the federated query processing is an important research topic in both theory and practice. Studies over the past decades have been working on optimising the performance of federated query processing. The performance consists of the execution time (Görlitz and Staab, 2011; Saleem et al., 2018; Endris et al., 2017), completeness of query answers (Aranda and Polleres, 2014), communication times (Deshpande and Hellerstein) and the number of outgoing sub-queries (Saleem et al.), in the cases of a lack of information about data sources (Charalambidis et al., 2015; Quilitz and Leser, 2008), data duplication (Saleem et al.; Montoya et al., 2015; Minier et al., 2018) and a wealth of data sources (Hasnain et al., 2016; Brickley et al., 2019). In the field of processing a federated SPARQL query, Saleem et al. (2017) and Rakhmawati et al. (2013) provided a summary and comparison of well-known federated query engines and frameworks using Schmidt et al. (2011) and Schmidt et al. as the benchmark for performance comparison.

Although runtime is a common performance of a query engine that its users care about, there are other optimisation goals to achieve when a query engine works for a data marketplace. With respect to the data requirements from buyers, the existing approaches

of the optimisation of federated query processing face the challenge of reducing the prices charged for collecting results from federated sources whose data can be priced by different pricing functions, while maximising the utility under the preferences/priorities of buyers. The up to date research on query optimisation has not considered this challenge yet.

Li et al. (2015) first introduced the price of data into federated SQL query processing and proposed a system, *PayLess*, to reduce the cost of the query process. It considered the challenges that the query engine has limited access to data sources and that the cost of intermediate retrieved data can be reduced by bind join. *PayLess* implemented a solution that reduced the amount of retrieved intermediate data with respect to the pricing function of data units in a data market where the pricing function is limited to price-per-unit. The drawback of *PayLess* is that it lacks concerns about the utility of query answers in terms of the preferences/priorities of buyers. This work overcomes the above challenge and improves the method of query processing and optimisation with respect to the price and value of data, as well as the budget of buyers.

2.5 Data Trade Deals

After collecting data from data sources, and before reaching a deal with buyers, a marketplace needs to make decisions about the data to return to buyers for a deal.

- Decision/Suggestion: this is the final phrase for a marketplace to decide how to sell data to buyers with respect to their requirements. It uses a target function to describe a buyer's preferences and budget, and also to rank the possible selections of answers in order to guide the buyer's decision making. Similar to Multiple Attribution Decision Making (Ding et al., 2016; Shivakumar et al., 2013), the factors within a target function are probably conflicting. For instance, data that satisfies the preferences/priorities could exceed the budget.
- Purchase Allocation: this undertakes the selection of a subset of a complete query answer for a deal with buyers following the decision or suggestion made by the marketplace in the above step.

Table 2.3 shows a summary of the literature relevant to the above component. The columns relate to the individual components, and the cells in each column indicate the method(s) that the cited study applied for the corresponding components. Li and Miklau (2012) proposed a framework that balances the price and privacy of personal data. Dong et al. (2012) suggested data buyers use the theory of marginalism to get the highest profits when purchasing data. *Bid&Plan* is a model proposed by Zollinger et al. (2012) that auctions the desired data. The auction allows sellers to generate a plan that satisfies the constraints of buyers, such as the price and quality of the data, and buyers to bid a price on it. On top of conventional databases or data networks, Dustdar et al.

(2012) introduced the concept of a quality-aware data integration service. Note that the component of purchase allocation has only been considered by Grubenmann et al. (2018a). They proposed a data marketplace, FedMark, which enables buyers to choose a subset of a complete query answer within their budget by introducing the concepts of allocation rules.

2.6 Performance Criteria

Regarding the discrete goals of data markets, the applied metrics to measure and evaluate the efficiency of a data market in performing data trades are diverse. We here categorise the goals of the existing markets, and then list the associated evaluation measurements.

- **Sellers' Welfare:** this is the major concern in the existing markets. There are two main elements of sellers' welfare: the protection of data privacy (Li et al., 2017; Nget et al., 2017) and revenue (Deep and Koutris, 2017b).
- **Buyers' Welfare:** this is the value that buyers obtain with respect to the payments they make. For the purpose of protecting or even increasing the welfare of buyers, Dong et al. (2012) evaluated their work by measuring the marginal cost of data purchases. Grubenmann et al. (2018a) and Stahl and Vossen (2016) allowed buyers to purchase according to their specific data quality demands.
- **Outcome Balance:** this aims at achieving a balance between the welfare of buyers and sellers. Ding et al. (2015) and Stahl and Vossen (2015) proposed data markets to balance the price and quality of data, whereas Moor et al. (2019) demonstrated the relationship between the data value that buyers obtain and the profits of data sellers or marketplaces. To this end, they measured the value and welfare of both buyers and sellers.

For the welfare of sellers, the essential metrics applied in the existing studies are the profit (revenue) and privacy, as Table 2.4 described. The welfare of buyers may be assessed in respect to the accuracy, completeness of data sources and the utility of their purchases. Due to the aim of this work, which is to fill in the gap of trading data with awareness of the diverse and autonomously set data pricing functions and the budget and utility constraints of purchases in data markets, our evaluation of a data market focus on its performance in assisting buyers to make purchases within their budget as well as to maximise the utility of a purchase.

TABLE 2.4: Evaluation Measurements for Welfare of Sellers and Buyers

	Metric	Description (Zaveri et al., 2015; Moor et al., 2019; Li et al., 2017)
Sellers Welfare		
	Profit	the degree of surplus of sellers or marketplaces in trading data
	Privacy	the degree of privacy loss of data sellers
Buyers Welfare		
	Accuracy	the degree to which the data values correctly represent the real world facts
	Completeness	the degree to which all the required information is present in a particular data source.
	Utility	the difference between the value of the data to the price paid by buyers for collecting the data (Riahi et al., 2013)

2.7 Existing Research Gap

The reviewed literature has offered solutions for the circumstances wherein sellers and buyers put full trust in a marketplace that takes charge of the data sources and services, data requirements and data search. However, the current research left gaps in the field of data trading. The existing designs of data markets limit the potential the pricing strategies. They either set up specific views or data points for sellers to assign prices, or assign the same price to all the data. This is not flexible enough for all the sellers who know the differences in the quality, value and costs of data collection and curation. Only limited research has been conducted on letting sellers use different pricing functions in one data marketplace.

In addition, various marketplaces and independent sellers who do not collaborate with any marketplace have not been considered together in the design of a data market. The current policy for sharing revenue among the sellers in a data marketplace assumed that all the sellers trust the marketplace and are willing to share the revenue gained from selling their data overlaps equally. However, this policy faces challenges. For instance, if one competing seller claimed that its data piece has been essential in the business and deserves a higher revenue, what would happen to other sellers' profits and buyers, especially when the data is duplicated in multiple data sources with different prices? In the worst case scenario, what if the seller threatened to quit since this would destroy the business in the market? Furthermore, where should buyers chase the remaining data sources to buy the pieces they need?

Moreover, the economic expectations of buyers, i.e. their budgets, has only been considered in the existing studies by means of the simple setting of data prices. Recall the component of purchase allocation, which indicates that the buyers can customise query answers regarding their budgets and preferences. This has only been considered by Grubenmann et al. (2018a). They proposed a data marketplace, FedMark, that enables

buyers to choose a subset of a complete query answer within their budget by introducing the concepts of allocation rules. The allocation rules can maximise the utility of a subset of query answers according to buyers' preferences/priorities as well. FedMark presented Integer Programming allocation and greedy allocation rules for implementation. However, the basis to apply the two allocation rules in FedMark (Grubenmann et al., 2018a) is that the marketplace is limited to apply only free and price-per-unit pricing strategies. This limits their allocation rules, i.e. the rules do not work when the price of data units is calculated by flat rate and freemium pricing strategies. Enabling diverse pricing strategies poses a new challenge for buyers making data purchases from massive data sources, especially with a budget constraint.

Furthermore, the data search in the existing data markets has not considered the prices of data. The state-of-the-art methods for searching data over massive data sources have only sought to improve the execution speed, i.e. the runtime. To the best of our knowledge, up to the date of writing the present work, price-based query optimisation is a blank area. The study that has the best possibility of improving execution speed as well as the prices of returned results is CostFed (Saleem et al., 2018), since it considered using different types of the join process to minimise the intermediate results and has the best performance compared to other approaches at that time. It defined the index that summarises the selected information of data sources, introduced a source selection algorithm based on the index and a greedy algorithm for query planning that approximates the lowest join cardinality, which contributes to reducing the execution time.

2.8 Summary

This chapter has attempted to provide an overview of the previous studies in the field of data markets. The overview is based on the architecture of a data market which abstracts the basic components considered in theory and practice to date. The architecture also presents the process of data trading among three participants: data buyers, data sellers and data marketplaces, which have data requirements, data sources and data searching ability, respectively. In addition, we discuss the organisational structures of data market implementations. For the concerns of this work, this chapter further describes the data sources and services, data requirements, data searching and trade deals within the concept of data markets, and summarises the instantiation of each component in the current work. Through this review, we argue that previously published studies have left research gaps in the field of the data market. This thesis aims to improve the data market itself and the subsequent optimisations required in trading data.

Chapter 3

Methodology

Data has become a critical asset for organisations to create value by optimising processes, developing business cases and implementing solutions. In the same way as raw materials are required to produce physical products, the data needed to fuel digital products is also a product itself, leading to the emergence of data markets, where data sellers and buyers trade data for money (Balazinska et al., 2011). The existing work on data markets offers good solutions to price the requirements of data and allows data sellers to assign varied pricing functions to their data sources. However, despite being considered a critical aspect of data business, very few studies have focused on for the technical problem of meeting the commercial expectations of sellers and buyers, i.e. the problem of improving data trading by enabling diverse data pricing strategies in data markets and assisting buyers to purchase data within their budgets. Previous studies either have not dealt with the constraints or tend to let a marketplace be responsible for those concerns based on sellers' and buyers' trust in (Li et al., 2017; Grubenmann et al., 2017b): (1) fair transactions, that the marketplace will not set any sellers at a more advantageous stage over others; (2) fair revenue sharing, that all sellers get paid for what they contribute; and (3) fair price, that buyers will get the optimal results for their money. Instead of putting trust in a marketplace, this work explores the method of trading data in a data market and investigates technical solutions to satisfy sellers regarding their expectations of controlling their pricing functions, and thereby, revenue, and buyers regarding their ability to purchase query answers within their constraints.

This chapter explains the research questions which we focused on and describes the research approach to answer those questions: first, we describe a design of a data market that can meet the requirements of both sellers and buyers; next, we discuss the technical challenges for the satisfaction of the expectations in data trading; after that, we introduce algorithms to approximate the optimal solutions to the challenges and demonstrate their efficiency. Following that, we present an introduction of the methods which are adopted in this work, and then describe the design for an efficiency analysis of our proposed solutions. The chapter concludes with a summary.

3.1 Research Questions

Data marketplaces act as intermediaries or arbiters of the transactions between buyers and sellers (Stahl et al., 2016; Fernandez et al., 2020). In the relationship among buyers, sellers and marketplaces, an essential aspect is who the trusted party is and conducts the necessary steps in trading data. In the existing data markets, it is the marketplaces that retain the trust from sellers and buyers to:

- access data for sale;
- collect query answers while mashup data from multiple sellers is possible;
- set the price of query answers and the contribution of associated sellers to the answers;
- in the event that an answer can be derived from different sellers, make decisions about how to purchase answers;
- process the payment and share it among the involved sellers.

However, such a marketplace-centred design is inefficient for both buyers and sellers with respect to their omitted expectations. From the perspective of sellers, pricing data, or claiming revenue for their data contributions, is a major aspect of their business. Although marketplaces adopt various policies to protect sellers from arbitrage opportunities, as reviewed in Chapter 2, letting a marketplace price data and calculate the revenue share is not flexible enough for sellers to manage data and control their profits. On the other hand, buyers expect marketplaces to make decisions for their benefit in the case that query answers can be derived from multiple sellers that offer duplicated data. Yet, if marketplaces do so, they will either end up with a decrease in the revenue of some sellers or a decrease in the revenue of all sellers by splitting the payment equally. Both outcomes conflict with sellers' expectations of maximising revenue. Faced with this dilemma, current markets tend to ignore the above-mentioned expectations from buyers and the constraints they have, such as budget and data preferences. To make it easier for both buyers and sellers to trade data, we explore methods to remove impediments and investigate the question of how to trade data with an awareness of the diverse data pricing autonomously set by sellers and the budget and utility constraints for purchases set by buyers. Our research plan to answer this question is first to explore the design of markets, and then update that design to remove impediments that disallow sellers from pricing data or buyers from purchasing data that meet their specific requirements. This leads to the following three research questions.

3.1.1 Research Question 1

Question: How to allow buyers to purchase data under their constraints in existing data markets while sellers can autonomously price their data?

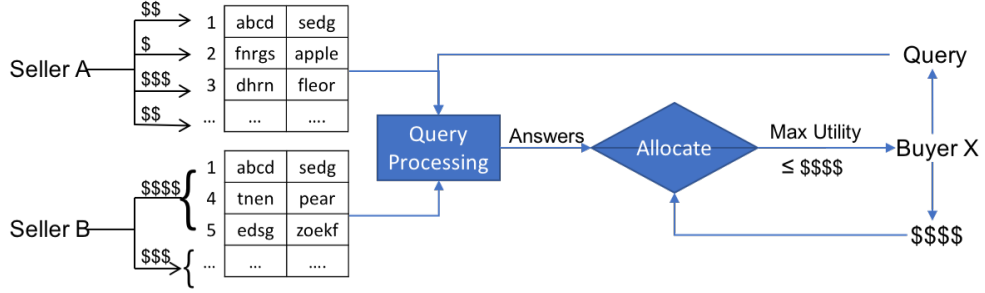


FIGURE 3.1: The problem of diverse pricing functions and buyer budget constraints within a federated data market.

First, through a comparison of centralised markets and federated markets, it is evident that federated data markets have advantages in terms of satisfying the expectations of buyers and sellers. Most centralised market designs expect sellers to trust a marketplace to set the prices that buyers will pay for query answers. This is useful in terms of disallowing arbitrage opportunities to buyers (Koutris et al., 2013). However, granting the data marketplace operator monopoly power over price settings comes with drawbacks. Buyers may be at the mercy of price changes set to maximise revenue for sellers or for the market operator. Sellers are not able to set their own pricing functions to compete with other providers or to compare their behaviour with the one set by the marketplace operator.

Federated data markets that allow sellers to set their pricing functions autonomously have been proposed by Grubenmann et al. (2018b,a). In this model, the sellers price data independently and the data market acts as an intermediary that takes the buyer's need, expressed as a query, has sellers' trust to access their data to compute the answer and compensates them for any data used to derive the answer purchased by the buyer. Regarding the expectations of buyers and sellers, the challenge of a federated marketplace lies in fulfilling the promise to the buyers and finding the optimal purchase for their money with respect to their preferences and budget constraints.

Therefore, we choose to build a federated data market where individual sellers set pricing functions for their data sources and where buyers can purchase data from the appropriate sources to answer their queries within their budgets. The above challenge is conceptually illustrated in Figure 3.1. After opening the optional pricing functions to sellers, the cases wherein different sellers price data in different ways (represented by the different number of '\$' symbols) arise. Each solution to a query is the output results of binding multiple data tuples purchased from different sellers and collecting matching items to the variables in the query. This indicates that individual solutions to a query can have different costs. When the budget of buyers is lower than the price of a complete query answer, the decision of choosing solutions to purchase is a problem for buyers (the decision of purchasing record 1 from Seller A or B). Under such a circumstance, if buyers trust a marketplace for the decisions, it is vital for the marketplace

to find the optimal allocation of solutions from the complete query answer to return to buyers for their benefit.

Previous work has only tackled this problem with respect to the pricing functions that assign a fixed price to each tuple (Grubenmann et al., 2018a), which does not allow the use of a large class of functions where the price of the data depends on the number of data units returned or accessed to derive solutions in the answer. Examples of these functions are those that model an economy of scale (a discount if the buyer buys more data) (Muschalle et al., 2012; Mehta et al., 2019); and a diseconomy of scale (more expensive if a buyer buys more data, e.g. as more information is revealed to the buyer) (Deep and Koutris, 2017b). However, when this class of functions, which we refer to as *access dependent* pricing functions (ADPFs), is allowed, the problem of computing a budget constrained allocation becomes more challenging, as the inclusion of a solution in an allocation may cause a cascade effect on the prices of other solutions already in the allocation.

We plan to provide a solution to the challenge raised by ADPF within our federated data market. The research approach is first to formalise the problem in a federated data market, and then to analyse the structure and complexity of the problem when ADPF is considered. After that, we divide the problem into two phases. The first is a compression phase aimed at minimising the price that needs to be paid for a set of data units. Since ADPFs price a set of data units differently from the total price of individual units, we split the required set of data units into a group of subsets, where each subset can be purchased from one data source, to find the minimum cost of the required set of data units, instead of purchasing data units one by one. Considering the possibility of data duplication among multiple data sources with different prices, *i.e.* the same data unit being sold by different sellers, we need to decide on a single source to purchase each *duplicated* unit for the goal of minimising the price to pay. Then, the allocation phase looks for a subset of the complete answer which costs no more than a budget with maximal utility.

In Chapter 4, we present a Cost Compression algorithm that purchases subsets of the input set of data units according to the pricing functions applied by sources, instead of selecting sources offering the lowest priced data units. We also implement two heuristic algorithms, Greedy and 3DDP, to approximate the optimal allocation while using the cost compression algorithm to minimise the cost of a subset of the complete answer based on the data units that derive the answer.

3.1.2 Research Question 2

Question: How to allow buyers to purchase data from multiple independent sellers and competing marketplaces under their purchase constraints?

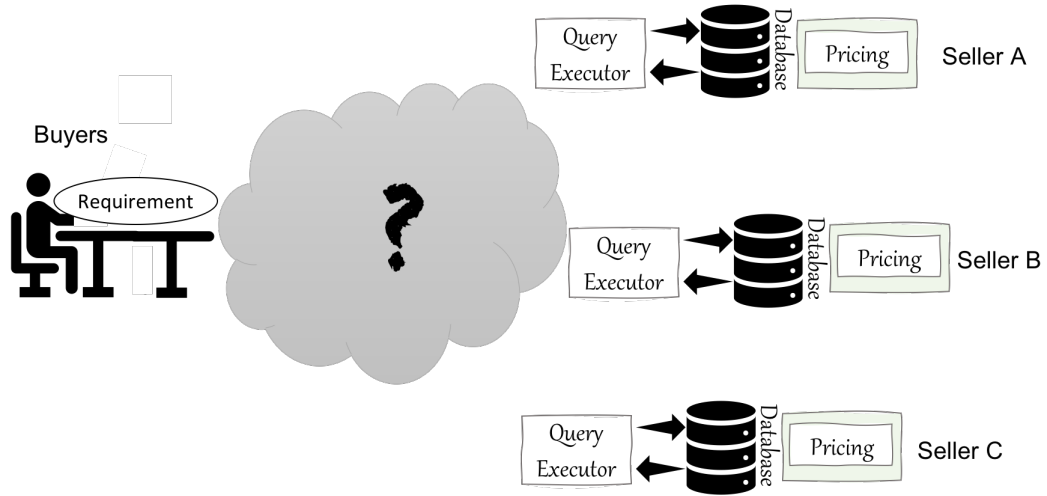


FIGURE 3.2: Problem of Letting Buyers and Sellers Directly Exchange Data

Recall the conflict a marketplace faces when it tries to satisfy the expectations of both sellers and buyers. When sellers doubt the equivalence of their income and contribution, they may stop offering free access to the marketplace, or quit, thereby leaving the marketplace or collaborators no choice but to face the risk of losing business. When buyers lose their interest in being victims of high bills and query answers that are not customised to their constraints, both sellers and marketplaces face the risk of losing business. In addition, the existing designs of data markets limit buyers to purchasing data only from one data marketplace. The problem of assisting buyers to make data purchases from multiple independent sellers or competing marketplaces has not been touched yet.

Accordingly, we plan to let buyers and sellers exchange or share data directly, which implies skipping the third party and releasing the required trust in the existing market designs. Figure 3.2 presents the initial design for this purpose. Distributed sellers autonomously manage and price their data and maintain a query executor for forthcoming queries. They can be as small as individuals trading their personal data or as big as a union of data sources, like an independent marketplace with its sellers. In the meantime, buyers have data requirements on the table and keep the right to decide where their money goes and how to collect data. This design eases the arduous trust in data markets, and it enables sellers to receive data revenue from buyers immediately after providing data. However, the remaining problem of this design is how to match sellers and buyers as well as process a data trade. These issues are solved by improving the above-mentioned design into a new data market architecture, Free Market. Free Market leaves query processing to sellers, while in the meantime letting buyers make purchase decisions, instead of using a centralised query engine like the existing markets. Free Market is designed to bring sellers and buyers together by way of announcements from a marketplace and a local query engine that works for buyers. The announcement from a marketplace provides buyers with the information about

the data sources available in the market, which is the basic method to propagate the updates of data sources (Dustdar et al., 2012). From the buyers' perspective, it is necessary but challenging to conduct cost-effective queries over a huge amount of available data sources in a data market. The local query engine will allow buyers to set up constraints for query processing and support buyers to purchase the desired answers from the market.

Chapter 5 details our design of a free market. As Grubenmann et al. (2017a) and Heling and Acosta (2020) claimed, without knowing specific data or free access to data sources, it is impossible to gain relevant knowledge of a query, not to mention optimisation. This means restricted settings, such as no open information of data sources, are obstacles to selling data to buyers when they have obvious difficulty querying and paying. For this reason, the public catalogue is designed to include three types of information about a data source: (1) a summary of metadata and statistics information; (2) the query interface for buyers to access the data; (3) the pricing scheme that allows buyers to estimate the costs of their queries. The local query engine in a free market processes queries differently from the existing query engines. Specifically, it optimises queries with an awareness of data prices and buyers' preferences and budget constraints. However, the existing studies of query optimisation have not considered the data prices and budgets yet, but rather aimed at reducing the execution time so as to execute queries time-efficiently (Saleem et al., 2017; Rakhmawati et al., 2013). It is challenging, however, to optimise queries before executing them. The reason is that precisely estimating the results of each step of the query processes remains a problem that will lead to the uncertainty in cost estimation, especially when the processes involve the procedure of join and union. Applying current time-efficient optimisations is a choice for buyers to optimise a query, but is there an option to save money? It is still unknown whether time-efficient optimisation methodologies also work money-efficiently, especially when the pricing functions of data sources are different.

Previous work (Li et al., 2015) has investigated the problem of query optimisation with an awareness of the data prices from the aspect of reducing the amount of intermediate retrieved data in the query process in order to pay less, and they implemented a system, PayLess. To find a query plan that can reduce the cost of query answers, PayLess Li et al. (2015) reduced the search space of query plans by proving that the optimal plan is a left-deep plan. It then applied the semantic query rewriting technique (Dar et al., 1996) to decompose a query by finding a union of views of data, *bounding box*, which covers all the data regions that buyers do not have at a minimum price. Finally it carried out the bind join of those views as the minimum cost data access plan. PayLess offers buyers a good orientation to save money. However, PayLess assumes that there is a linear relationship between the number of intermediate results and their price, yet this assumption is invalidated when access-dependent pricing functions (ADPF) are used for pricing data by sellers. The price variation of data sources from different sellers is

also not considered in PayLess despite this being an expectation from sellers of open options for pricing functions in the data market in our work. Furthermore, PayLess does not reveal the impact of the data price on query optimisation, despite the fact that buyers will be expecting a price-sensitive query planning algorithm which changes query plans according to the prices of data. In addition, no customisation of query answers is considered for the purpose of purchasing data within budget constraints.

To solve the above-mentioned problem, we propose price-based query planning approaches within a design of a local query engine and satisfy the buyers' requirements to reduce the costs of query answers. We apply a graph structure and its spanning tree to model query and query plans, assign the costs of collecting intermediate results as the weight of edges of the graphs and, thereby, a minimum spanning tree of the weighted graph is a query plan with the lowest costs. Further details about this model are explained in Chapter 6. Chapter 7 brings the budget constraint into consideration and casts the problem of query optimisation with a budget constraint into a weight-cost constraint minimum spanning tree problem.

3.1.3 Research Question 3

Question: What are the differences between different types of data markets in terms of their assistance to sellers and buyers?

In order to draw overall answers to our original questions and review all of our work together, we plan to answer the above research question through a comparison of the data markets: general markets, data emporium and the free market. We first compare their structure in terms of the data trading process, then characterise the market services for sellers:

- Data Management: What data management services does a market offer?
- Trust: What authorisation does a market require from a seller?
- Pricing: How does a market suggest calculating data prices?
- Dependency: What is the relationship among sellers which a market defines?

and services for buyers:

- Budget: Does a market allow buyers to customise their financial constraints?
- Efficiency: What service does a market offer buyers for their requirements?
- Searching Process: What operations does a market take to collect data for buyers?
- Settlement: Will a market take the payment on behalf of sellers from buyers?

for a further comparison in Chapter 8.

3.2 Foundations

3.2.1 Format of Data and Query

We plan to use RDF (Cyganiak et al., 2014) data and SPARQL (Harris et al., 2013) query for our implementation and evaluation. Our later explanation of this work is based on the following notation and hypothesis.

- RDF is the W3C recommendation for representing information in the Web. The basic unit of RDF is *triple* $t = (s, p, o)$, where s is a subject, p a predicate and o an object, to denote information entity s owns a property p with a value o . s, p, o in $t = (s, p, o)$ may be URIs, blank nodes or literal values corresponding to the information. For the sake of brevity, let U represent the set of all valid RDF terms.
- SPARQL is the W3C recommendation query language for RDF data. A SPARQL query, q , comprises patterns that match RDF triples called *triple patterns*, tp . A triple pattern $tp = (x, y, z)$ is written as the subject, predicate and object following the conventional order of RDF triples. The three terms x, y, z in $tp = (x, y, z)$ are either variables or valid RDF terms in U . We say a triple pattern $tp = (x, y, z)$ matches a triple $t = (s, p, o)$ when x, y, z either are variables or equal to s, p, o , respectively.

Hypothesis. A sub-query decomposed from a SPARQL query is constructed by a *SELECT* clause and a *WHERE* clause (W3C), where the *WHERE* clause provides one, and only one, triple pattern from the query and the *SELECT* clause identifies all the variables in the triple pattern.

3.2.2 Related Algorithms

To improve the data market, the essential question in this work is to solve the allocation problem and the query planning problem with the concerns of data prices and budgets. These problems belong to the category of optimisation problems and are similar to the knapsack problem and graph theory which have been studied for decades. Therefore, we plan to adopt the existing algorithms for these optimisation problems, and then propose adaptive algorithms to solve the challenges brought by the data prices and budgets. This section introduces the dynamic programming and greedy algorithm which we adopt to solve the allocation problem, the minimum spanning tree problem and algorithms which we use to model query plans to estimate their costs.

- Dynamic Programming: this is the model of a dynamic system that deals with sequential decision making processes under the guidance of a decision maker

(Neumann, 1993). It solves problems by dividing the problem into overlap sub-problems and recursively combining solutions to the sub-problems into the solution to the original problem. According to Bellman’s optimal policy (Bellman and Lee, 1978), if a problem exhibits the optimal substructure, dynamic programming will find an optimal solution to the problem (Dijkstra et al., 1959).

- Greedy Algorithm: this is one of the algorithms for optimisation problems. It solves problems by always finding the optimal intermediate solutions at each stage of the sequential decision making process (Cormen et al., 2009). This means that the decisions are locally optimal in the hope that the combination of locally optimal solutions leads to a globally optimal solution. Therefore, greedy algorithms do not always guarantee optimal solutions.
- Minimum Spanning Tree: this is the spanning tree of a connected, undirected, weighted graph which has a minimum cost with respect to the weight of either nodes or edges in the graph. The problem of determining a minimum spanning tree is the minimum spanning tree (MST) problem. Existing well-known MST algorithms are *Boruvka* (Boruvka, 1926), *Prim* (Prim, 1957) and *Kruskal* (Kruskal, 1956), which are all greedy algorithms and are proved for finding an optimum solution in cases of constant weights or costs of edges. Aggarwal et al. (1982) first introduced the weight-cost constraint minimal spanning tree problem and proved its weak NP-hardness. The approximate solutions are from Ravi and Goemans and Henn (2007).

3.3 Evaluation Design

To measure the efficiency of our solutions regarding the research questions, we evaluate their performance in a synthetic data market. The data market includes a query engine that executes queries on top of data sources from the FedBench (Schmidt et al., 2011) benchmark, accepts various settings to price data and embeds our models and algorithms and the state-of-the-art approaches.

- Equipment: we use a Red Hat Enterprise Linux 7 (RHEL7) virtual machine with four CPU Cores and 32GiB RAM and 200GiB storage for the experiments.
- Benchmark: we adapt the well-used FedBench (Schmidt et al., 2011) data sources and queries in our experiment. There are 9 data sources: ChEBI, DBpedia, DrugBank, KEGG, Geonames, Jamendo, LinkedMDB, NY Times and SW Dog Food as well as 25 queries.
- Pricing functions: We specify the 4 categories of *Data-based Formula* in Chapter 2 as functions in Table 3.1 and apply them in our experiments.

The evaluation section in the following chapters will further explain the details of generating the data requirements with budget and utility constraints and sellers with

TABLE 3.1: Basic Pricing Functions. T denotes a set of data units, $o(T)$ is the number of units in T , P is the flat rate price of an entire data source, and R is the number of units free of charge.

Name	Description	Notation	Function	Example
Free	The price of any data unit is 0.	FPF	$FPF(T) = 0$	Dbpedia
Price-per-unit	Each data unit has a specific price.	PTPF	$UBPF(T) = \sum_{t \in T} p(t)$. where $p(t)$ is the price of data unit t	Vodafone
Flat	Any data unit can be used by paying a fixed price.	FlatPF	$FlatPF[P](T) = P$	iTunes
Freemium	Free up to R data units, but premium data is set with a flat price.	FMPF	$FMPF[P, R](T) = \begin{cases} 0 & o(T) \leq R \\ P & o(T) > R \end{cases}$	Photable

or without a database, access interface and its pricing function based on the above-mentioned elements. The evaluation shows that our approaches approach 85.7% optimal allocations and reduce average cost by 75% compared to the state-of-the-art query planning algorithms.

3.4 Summary

This chapter has described the methodologies in our investigation of the problem of optimising data trade by enabling diverse data pricing strategies in data markets and assisting buyers to purchase data within their budgets. We argue that, as the main participators in data trading, sellers' expectations of controlling their pricing functions, thereby revenue, and buyers' expectations regarding their ability to purchase query answers under their constraints, have not been satisfied in previous studies. To close this gap, we plan first to explore the methods that allow buyers to purchase data under their constraints in existing data markets while sellers can price their data autonomously, and then we intend to design a data market where buyers can purchase data from multiple independent sellers (i.e. individual sellers or competing marketplaces) under their purchase constraints and, lastly, we compare our work with the existing studies with respect to the satisfaction of buyers and sellers from the services of the data markets. Following the research plan, we introduce the data structure and basic algorithms applied in our research as well as the design for the evaluations in our work.

Chapter 4

Purchase Allocation for Limited Budgets within the Data Emporium

As an alternative to the centralised data markets, federated markets enable sellers to set pricing functions for their data autonomously and sell affordable subsets of a query answer to meet the limited budget of a buyer without putting much trust in a marketplace. Compared to the centralised data markets, where the marketplace sets the price of query answers, the challenge of a federated marketplace is to fulfil the promise to the buyers and to find the best purchase for their money: Given that sellers price their data sources in different methods, find a way to purchase data and allocate answers to buyers' queries so as to maximise the utility of the purchased data within their limited budgets, especially when the price of the data depends on the number of tuples returned or accessed to derive solutions in the answer.

In this section, we first explain a federated market, Data Emporium, which allows sellers to apply different types of pricing functions, generalise the problem of purchase allocation with budget constraints with an analysis of its complexity and specify the new challenge arising from ADPFs. After that, we introduce our two-step approach to solve the problem: the cost compression algorithm to minimise the cost of intermediate allocations, and the Greedy and 3DDP heuristics to approximate the optimal allocation. Then we present our experimental evaluation of their performance and a summary of this chapter.

4.1 Data Emporium

We create a federated marketplace, Data Emporium, which contains (1) a set of data sources from sellers, D , (2) a federated query engine that has the trust of the sellers to access data sources, E , and (3) a set of pricing functions used by data sources to

TABLE 4.1: Example Data Sources and Query Answers

(a) Data Sources with Tuples and Prices

Data Sources	Data Tuples	Price
d_1	t_1	0.7
	t_2	0.3
	t_3	0.2
d_2	t_3	FlatPF[1.5]
	t_4	
	t_5	

(b) Query Answers

The diagram illustrates the mapping from data sources to query answers. On the left, data sources d_1 and d_2 (represented by cylinders) are shown. Arrows indicate the flow of data to a result set RT (represented by blue rounded rectangles). d_1 maps to t_1 and t_2 , while d_2 maps to t_1 , t_3 , and t_4 . The tuples in RT are then mapped to query answers s_1, s_2, s_3 (represented by orange rounded rectangles). Specifically, t_1 and t_2 map to s_1 , t_1 and t_5 map to s_2 , and t_3 and t_4 map to s_3 . Vertical dashed lines separate the data sources, the result set, and the query answers.

calculate the price of a set of data tuples, PF . This section presents the model of data sources and query answers collected by the query engine and calculations of prices in Data Emporium. Notice that we focus on the problem of purchasing query answers from multiple data sources at a minimum price. Thus, the details of federated query execution, payment process and trust issues, etc., will not be discussed in this chapter, but please refer to Chapter 2 for a general discussion.

4.1.1 Data Sources

For generalisation, instead of specifying a database and a corresponding query language, our description is built on the basic understanding of the join process and the well-known notations of *tables*, *tuples*, *rows* and *triples* from both the relational database field (Abiteboul et al., 1995; Das et al., 2012; Garcia-Molina, 2009) and Linked Data (Berners-Lee, 2009; Cyganiak et al., 2014; Harris et al., 2013).

Data sources come in a range of structures and format, including relational and RDF. We model the basic unit of data as a tuple, t . For instance, in a relational database, t is a single row in a table. Likewise, a triple in the form of RDF (Cyganiak et al., 2014) is notated as t . A data source, d , is a collection of tuples. The federated marketplace allows data duplication, which means the same tuple may exist in multiple data sources. Table 4.1(a) displays two example data sources, and the duplicated tuple between them is t_3 .

4.1.2 Query Answer Derivation

When a buyer asks a query, the federated query engine, E , computes the complete answer $S = \{s_1, s_2, \dots, s_k\}$ to the query, where s denotes a single solution to the query. For instance, the buyer has a SPARQL query - *SELECT * WHERE {?student rdf:studyIn university:UOS, ?student rdf:nationality ?country.}* - to enquiry about the nationality of the students at the University of Southampton. There is a dataset about the Electronics

and Computer School (ECS) that consists of the following two tuples - $\langle \text{student:Miya} \text{ rdf:studyIn university:UOS.} \rangle$, $\langle \text{student:Miya} \text{ rdf:nationality country:China.} \rangle$. Hence, a solution to the buyer's query is $\text{student:Miya, country:China}$. Note that if this dataset has more relevant data tuples, the buyer can get more solutions in the final complete answer to the query.

To abstract the query execution process in varied query languages, we restrict queries to the Select-Project-Join type and consider solutions as the result of the joining of tuples from one or more data sources, annotated with their provenance. We denote a solution as $s : \bowtie_{i=0}^x t_i^{d_i}$, where $t_i^{d_i}$ represents that tuple t_i is from data source d_i , and s is the output result of bind join x tuples. In the above example, the solution is the result after joining the two tuples by the same subject - student:Miya and both tuples are from the dataset about ECS.

Due to the possibility of data duplication over multiple data sources, the above-mentioned process can produce the same solution by collecting the same tuples from different sources. We call the set of tuples derived into a solution, s , the *relevant tuple set* of s , notated as $RT(s)$. Note that provenance annotations (Green et al., 2007a) allow us to distinguish the data sources of tuples, especially when the same tuple is provided by different data sources.

Running Example Table 4.1(b) illustrates a scenario of a query with three solutions derived from the two data sources in Table 4.1(a). Following our notation, the three solutions are:

- $s_1 : t_1^{d_1} \bowtie t_2^{d_1}$, with $RT(s_1) = \{t_1, t_2\}$;
- $s_2 : t_1^{d_1} \bowtie t_5^{d_2}$, with $RT(s_2) = \{t_1, t_5\}$;
- $s_3 : t_3^{d_1} \bowtie t_3^{d_2} \bowtie t_4^{d_2}$, with $RT(s_3) = \{t_3, t_4\}$.

This example showcases the possibility of overlaps among the relevant tuple sets of solutions and the multiple data sources offering the same tuples in the derivation of solutions, which can be seen in the case of solution s_1 with s_2 , wherein tuple t_1 is required to derive both solutions, and of solution s_3 whose relevant tuple t_3 is offered by d_1 and d_2 .

4.1.3 Price of Data and Query Answers

When buyers decide to purchase answers to their queries, the data market calculates the prices of data according to PF . Without a loss of generality, we assume that sellers assign each of their source d a pricing function $PF[d]()$, which takes a tuple set as input and calculates a non-negative value as the price of purchasing the set from d . Note that multiple data sources may sell the same tuple set at different prices.

To compute the cost of the complete answer, the marketplace needs first to compute the cost of a single solution. To do so, the sources and prices of all the relevant tuples need to be considered. When a relevant tuple is offered by multiple data sources, like t_3 in the running example, the buyer should only pay once; that is, only one source should be selected. Let $sub(s, d)$ denote the subset of $RT(s)$ that is chosen to be purchased from data source d , and then the cost of s is

$$c(s) = \sum_{d \in D} PF[d](sub(s, d)). \quad (4.1)$$

However, the price of the complete answer, S , is not $\sum_{s \in S} c(s)$, since the $RT(s)$ of different solutions may overlap (such as s_1 and s_2 in the above example). Adding the costs of individual solutions can result in overpaying for the same tuple, which would be unfair for buyers. Consequently, the calculation of the price of S should be made based on tuples relevant to the whole set S . Note that $RTA(S) = \cup_{s \in S} RT(s)$, as the relevant tuple set of the query answer S . We denote $SUB(RTA(S), d)$ as the subset of $RTA(S)$ that is chosen to purchase from data source d by markets.

In the running example, $S = \{s_1, s_2, s_3\}$, and $RTA(S) = \{t_1, t_2, t_3, t_4, t_5\}$. Thus, the cost of S is

$$C(S) = \sum_{d \in D} PF[d](SUB(RTA(S), d)). \quad (4.2)$$

For the running example, a possible way to purchase tuples that derive solutions in S is to buy two subsets from d_1 and d_2 together: $SUB(RTA(S), d_1) = \{t_1, t_2, t_3\}$, and $SUB(RTA(S), d_2) = \{t_4, t_5\}$.

4.2 Purchase Allocation

When a buyer is constrained by a budget that is lower than the cost of a complete answer, they may want to purchase a subset they can afford. Data Emporium will help buyers by allocating solutions and purchasing a subset of the complete answer within their budget, leading to the problem of *Purchase Allocation* (PA). This section generalises the PA problem and describes the challenge that access-dependent pricing functions (ADPFs) bring to it.

4.2.1 The PA Problem

We assume that buyers provide the marketplace with a valuation, $v(s)$, to specify the utilities of solutions and guide the purchase allocation. The utility indicates the importance buyers put on solutions or buyers' preferences (Zaveri et al., 2015; Dustdar et al., 2012; Thakkar et al., 2016; Truong and Dustdar, 2009). For instance, for a query that

asks about data from different countries, a buyer could quantify the utility of solutions to signal that data from China is more valuable for them, and it should be privileged in the allocation. Alternatively, $v(s)$ can be equivalent over all the solutions to signal that there is no particular preference. The problem of defining and computing $v(s)$ is beyond the scope of the present work.

The PA problem:

INSTANCE: A finite set of k solutions, $S = \{s_1, s_2, \dots, s_k\}$; a utility function $v(s) \in \mathbb{R}^+$; a finite set of data sources, $D = \{d_1, d_2, \dots, d_m\}$, a function, $PF[d](T)$, for each source $d \in D$ to calculate the price of tuple set T ; a relevant tuple set, $RT(s)$, for each solution $s \in S$; a budget bound, B ; an allocation Λ is a subset of S , and its full relevant triple set $RTA(\Lambda)$, $RTA(\Lambda) = \bigcup_{s \in \Lambda} RT(s)$.

QUESTION: Find the allocation having maximum utility and decide on the sources to purchase the relevant tuples with an overall cost no greater than B .

The settings of the pricing functions affect the difficulty of the PA problem. Recall that the same tuple can be assigned with varied prices from different sources, and that $C(S) \neq \sum_{s \in S} c(s)$. The cost of each solution is calculated from the pricing functions of the sources selected to purchase its relevant tuples, and the cost of an allocation depends on its full relevant triple set. With respect to the duplication of tuples assigned with varied prices in multiple sources, the essential question of PA is to search for a partition of $RTA(\Lambda)$ where each subset in the partition can be purchased from a data source based on the provenance that distinguishes sources of the duplication, while minimising the total cost of the partition according to the pricing functions of each source. When the price of each tuple is a non-negative constant, it is possible to get the partition by picking tuples from the data sources in D that offer them at the lowest price.

4.2.2 ADPF

Through observations of the real world, it is evident that there are pricing functions that set data prices according to the number of bought tuples instead of having fixed prices for every specific data tuple (Stahl et al., 2017). Below, we define these pricing functions as access-dependent pricing functions.

Definition 4.1. Access-Dependent Pricing Function (ADPF): a pricing function $f(T)$ is access-dependent if, and only if, tuple sets T_1 and T_2 exist such that $f(T_1) + f(T_2) \neq f(T_1 \cup T_2)$.

Two notable examples of ADPF are flat-rate and freemium. The flat-rate pricing function assigns all the required tuple sets with the same price, notated as $FlatPF[P](T) = P, \forall T$, where P is the flat-rate price. The freemium pricing function offers a limited

number of free tuples but any larger tuple set with a greater number charges a price, notated as $FMPF[P, N](T) = 0, \text{ if } |T| \leq N; FMPF[P, N](T) = P, \text{ if } |T| > N$, where N is the threshold for the free quota.

When data sources use ADPF, the lowest price of a data tuple is not a constant. Purchasing tuples from the data sources with the lowest price does not guarantee the minimal cost of a tuple set. The purchase decision, therefore, need to find the sources to minimise the overall cost of a tuple set over multiple data sources. Furthermore, the cost of adding a solution to an allocation is affected by solutions in the allocation, which is due to the possibility of overlap between their relevant tuple sets. ADPF amplifies the new challenge to the essential partition question of the PA problem.

For the running example, data source d_2 in Table 4.1(a) applies the pricing function $FlatPF[1.5]$, which indicates that the cost of a tuple set with any tuple from d_2 is 1.5. Assume a buyer purchases an allocation $\Lambda = \{s_2, s_3\}$ with its $RTA = \{t_1, t_3, t_4, t_5\}$. If the buyer choose data sources to purchase tuples based on the price of individual tuples, the choice is to purchase t_3 from d_1 rather than d_2 , since $PF[d_1](\{t_3\}) = 0.2$, $PF[d_2](\{t_3\}) = 1.5$, and $0.2 < 1.5$. However, if one takes into account that there are another two tuples, t_4 and t_5 , that only exist in d_2 , and if the choice was to purchase them from d_2 , and then purchasing t_3 from d_2 along with t_4 and t_5 will reduce the cost of the allocation by 0.2 compared to purchasing t_3 from d_1 .

4.2.3 Complexity Analysis

Theorem 4.2. *The allocation problem (PA) is NP-hard.*

Proof. We prove that PA is NP-hard under the following simplifying assumptions: each tuple has a non-negative constant value as its price. Then, it is straightforward to get the lowest price of each tuple without considering the difficulty introduced by pricing functions.

We reduce a known NP-hard problem, the *Set-Union Knapsack Problem* (SUKP) (Goldschmidt et al., 1994) to the PA problem on the above assumption. Following the formalisation of SUKP in Kellerer et al. (2003), the SUKP problem is:

INSTANCE: A set of n items, N , $N = \{1, \dots, n\}$, a set of m elements, P , $P = \{1, \dots, m\}$ and a knapsack with a capacity of c . Each item j corresponds to a subset P_j of the element set P . Each item j has a non-negative profit, p_j , and the element i has non-negative weight w_i . The total weight of a collection of items is given by the total weight of the elements in the union of the items' element sets. QUESTION: Find the subset of the items with the maximal profit of a subset of the items and its total weight not exceeding the knapsack capacity.

Define an instance I of problem PA reduced from SUKP by:

$$\begin{aligned}
 S &= N, \\
 RTA(S) &= P, \\
 RT(s) &= P_j, \\
 v(s) &= p_j, \text{ where } s = j, s \in S, j \in N, \\
 PF[d](\{t\}) &= w_i, \text{ where } t = i, t \in RTA(S), i \in P, d \in D, \\
 &\text{and } d \text{ is the data source that has the cheapest price of } \{t\}, \\
 B &= c.
 \end{aligned}$$

We will explain that there is an answer for the instance I of the PA problem on the assumption of a maximal utility within the budget of B , if, and only if, there is a procedure to find the subset that satisfies the question of SUKP. Assume the subset Φ of N has a maximal profit without exceeding the capacity of c . Since the values of solutions in S and the budget are equal to the profits of items in N and the capacity, the allocation $\Lambda = \Phi$ has maximal value within the budget of B and answers the instance I of problem PA. And vice versa.

Using the reduction from SUKP to the PA problem establishes the simplified problem as NP-hard. Then, when introducing any other pricing functions (e.g. ADPFs), the PA problem is at least in NP-hard. \square

FedMark (Grubenmann et al., 2018a) provides a greedy algorithm and an integer programming solution for the PA problem when each data tuple has a fixed price. Unfortunately, these do not work when data sources use ADPF. Falling back to a brute-force search is unlikely to scale, as it has a complexity of $O(2^k \times m^n)$, where k is the size of the complete answer, m is the number of data sources and $n = |RTA(S)|$. In the next section, therefore, we propose two heuristic-based algorithms, Greedy and 3DDP, to approximate the optimal allocation.

4.3 Approach

Our approaches apply two phases—(1) compression and (2) allocation—to find the approximate optimal allocation of PA. First, a cost compression algorithm is proposed to find the partition of a relevant tuple sets so as to compress the cost of an allocation. The partition indicates the sources selected to purchase relevant tuples of the allocation at the lowest costs when sources have overlapping data and are assigned with different ADPFs. To approximate the optimal allocation, we then apply the two heuristic methods, greedy and dynamic programming.

4.3.1 Running Example

Recall the example detailed in Table 4.1. For the query q , a buyer is looking for an allocation within a budget of 2.5. The two data sources, d_1 and d_2 , price the tuple as shown in Table 4.1(a). The complete answer of q collected by a query engine is S , $S = \{s_1, s_2, s_3\}$, with the relevant tuple sets of solutions displayed in Section 4.1.2, and $RTA(S) = \{t_1, t_2, t_3, t_4, t_5\}$. Therefore, the question is to find an allocation of S without exceeding the budget of 2.5.

4.3.2 Cost Compression

With ADPF, choosing to buy individual tuples with the lowest prices does not guarantee the minimum cost of a set of tuples, yet minimising the cost of an allocation. Following from the example in Section 4.2.1, choosing a single tuple from a source using FlatPF incurs a cost equal to the flat-rate, which can be very expensive and impossible to be considered as the lowest price of the tuple. However, once this has been done, it can reduce the price of a set of tuples, potentially cutting the cost of an allocation. In the case of a source that uses FMPF, any individual tuple chosen below the free quota has no cost, but when a tuple set just exceeds the free quota, its cost will increase sharply, and the price of an allocation will also rise with it.

To calculate the lowest cost of an allocation Λ for purchasing $RTA(\Lambda)$ from overlapping data sources, a brute search approach needs to examine all m^n ways to select sources for $RTA(\Lambda)$, where m and n are the number of data sources in the market and tuples in the set, respectively. We approximate the minimal cost of $RTA(\Lambda)$ by a *Cost Compression* algorithm, $Zip(RTA(\Lambda))$. Note that the following discussion in this chapter only uses two pricing functions, *Flat* and *Freemium*, as the example of ADPFs, but the following introduced cost compression algorithm can work with any pricing functions in the family of ADPF.

The basic idea of cost compression is to purchase subsets of $RTA(\Lambda)$ instead of tuple by tuple. Algorithm 1 shows the pseudocode of our approach. Start with getting the subsets of $RTA(\Lambda)$ that each data source can offer (Lines 1-7). Calculate the costs of all the subsets (Lines 8-10). If there exists a data source that offers all tuples in $RTA(\Lambda)$ and charges a minimum price among all the data sources, then it is the ideal source to purchase $RTA(\Lambda)$ (Lines 11-12). Otherwise, first pick the tuples that exist in multiple data sources (Lines 15-19); then, sort the tuples by their minimum costs among the data sources in ascending order (Line 20); next, start an iteration of the tuples existed in multiple sources, and in each tuple, remove each tuple from a subset offered by a data source unless the cost of purchasing the tuple in the subset from the data source is lowest (Lines 21-33); finally, sum up the cost of each subset as the compressed cost of $RTA(\Lambda)$ (Lines 34-37).

Algorithm 1: Cost Compression, $Zip(RTA(\Lambda))$:

Input: $RTA(\Lambda)$, a set of tuples; PF , the pricing functions of data sources.**Output:** z , the compressed cost of purchasing $RTA(\Lambda)$

```

1 tupleSources, tupleSetPrice  $\leftarrow \emptyset$ ;
2 foreach  $t \in RTA(\Lambda)$  do
3   | sourceList  $\leftarrow$  data sources that offers  $t$ ;
4   | foreach  $d \in sourceList$  do
5   |   | tupleSources[ $d$ ].add( $t$ );
6   | end
7 end
8 foreach  $d \in tupleSources.Keys()$  do
9   | tupleSetPrice[ $d$ ] =  $PF[d](tupleSources[d])$ ;
10 end
11 if  $\exists d$ ,
    |  $tupleSources[d] = RTA(\Lambda) \ \&\& \ tupleSetPrice[d]$  is the lowest then
12   | return tupleSetPrice[ $d$ ];
13 else
14   | multiSourceTuples  $\leftarrow \emptyset$ ;
15   | foreach  $t \in RTA(\Lambda)$  do
16   |   | if  $t$  exists in multiple data sources then
17   |   |   | multiSourceTuples.add( $t$ );
18   |   | end
19   | end
20   | Sort (multiSourceTuples) by the minimum price of  $t$  over its sourceList in
    | ascending order;
21   | foreach  $t \in multiSourceTuples$  do
22   |   | reducedCost  $\leftarrow \emptyset$ ;
23   |   | sourceList  $\leftarrow$  data sources that offers  $t$ ;
24   |   | foreach  $d \in sourceList$  do
25   |   |   | tupleSet  $\leftarrow tupleSources[d] - \{t\}$ ;
26   |   |   | reducedCost[ $d$ ]  $\leftarrow$ 
    |   |   | tupleSetPrice[ $d$ ] -  $PF[d](tupleSet)$ ;
27   |   | end
28   |   |  $d_x \leftarrow$  reducedCost[ $d_x$ ] is the lowest;
29   |   | foreach  $d \in sourceList \ \&\& \ d \neq d_x$  do
30   |   |   | tupleSources[ $d$ ].remove( $t$ );
31   |   |   | tupleSetPrice[ $d$ ] =  $PF[d](tupleSources[d])$ ;
32   |   | end
33   | end
34   | foreach  $d \in tupleSetPrice.Keys()$  do
35   |   |  $z += tupleSetPrice[d]$ ;
36   | end
37   | return  $z$ ;
38 end

```

The cost compression algorithm avoids selecting sources for duplicated tuples that result in higher subset prices. It computes the compressed cost with computational complexity $O(n \log(n))$, where n is the number of tuples. The limitation of the algorithm is that it is not guaranteed to calculate the minimal cost. To our knowledge, however, the partition of $RTA(\Lambda)$ over ADPF is a new challenge, and the cost compression provides a linear approach to approximate the optimal answer. Later, in Chapter 4, the experimental results show that the cost compression assists the allocation algorithms to find about 85% optimal answers.

4.3.3 Greedy Algorithm

In a general case, a greedy heuristic would initialise each solution with the compressed cost of its $RT(s)$, sort them by utility/cost ratio and allocate a solution with the maximum utility/cost ratio until the cost of the allocation exceeds the budget. However, the general way to apply a greedy strategy is invalid for the PA problem.

First, the costs of the solutions are related when they share the same tuples in their relevant tuple sets, as we explained in Section 4.2.1. Second, in the presence of ADPF, allocating a new candidate solution into an allocation Λ and compressing the cost of $RTA(\Lambda)$ can result in selecting different sources for the relevant tuples of the solution, from the ones selected for the compressed cost of its $RT(s)$. This means that the cost of allocating s into an allocation Λ , is not equal to the compressed cost of its $RT(s)$. Therefore, sorting solutions by the rate of utility to their compressed cost will not yield the desired outcome.

To handle this issue, we need to allocate each candidate solution into a current allocation and compute the compressed cost of the attempted allocation. Thus, the greedy algorithm will allocate the candidate solution that maximises the utility/cost ratio of the attempted allocation. For the example in Section 4.3.1, Figure 4.1 illustrates the progress of the greedy algorithm. It starts by initialising the compressed cost of each of the candidate solutions to be allocated. Regarding the same utility over solutions, the greedy algorithm allocates solution s_1 . Then, it updates the compressed cost of attempted allocations for each of the remaining solutions and sorts them by their costs. After that, it makes a greedy selection among the attempted allocations, which is $\{s_1, s_2\}$. Next, it iterates the process of updating, sorting and then allocating. Finally, the algorithm will terminate when the cost of the allocation is greater than the budget, or when no candidate solution is left. The final allocation is $\{s_1, s_2, s_3\}$ with selected sources d_1 for $\{t_1, t_2\}$ and d_2 for $\{t_3, t_4, t_5\}$.

The greedy algorithm obtains its final allocation by iterating all the candidate solutions and applying the cost compression in each loop. Hence, its complexity is $O(n^2 \times k)$.

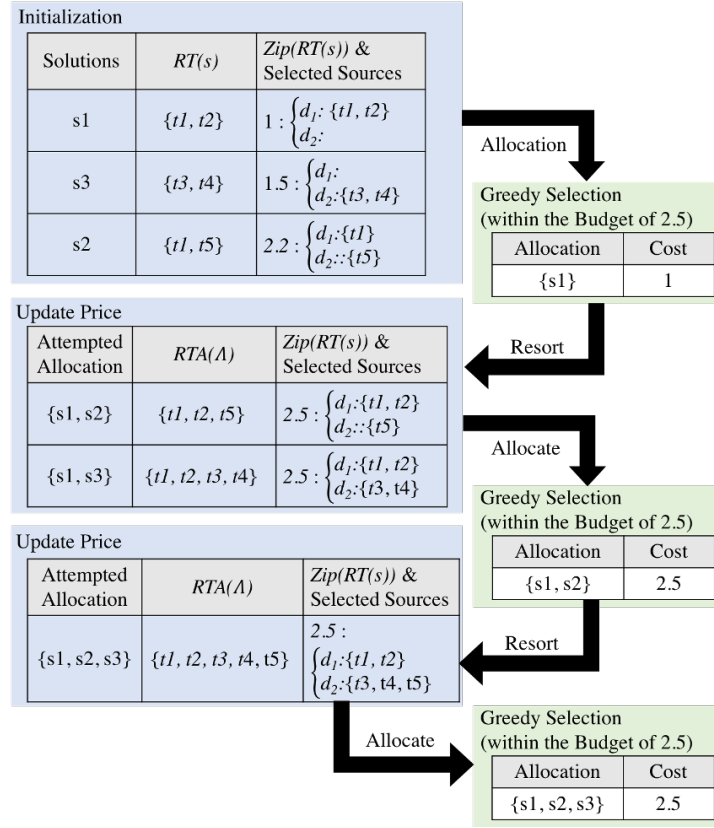


FIGURE 4.1: Process for Solving the Example Problem Using Greedy Algorithm

4.3.4 3DDP Heuristic

Our second heuristic algorithm, 3DDP, is based on dynamic programming. First, we split the PA problem into sub-problems by increasing the solution set from empty to the full set and a budget from 0 to the budget constraint. Let ϵ denote the increase step of budgets. In addition, we introduce an extra dimension to the solutions and budgets, in order to record the picked sources for purchasing tuples. Recall from Section 4.2.1 that the cost of an allocation is not determined by the sum of the costs of the allocated solutions, but rather by the sum of the prices assigned by the data sources for the tuples in their relevant set. Thus, the dimension is designed as a $(0,1)$ -matrix $DT = (a_{ij})_{m \times n}$, where m and n are the number of data sources in D and tuples in $RTA(\Lambda)$, respectively. $a_{ij} \in (0,1)$ represents the decision on purchasing tuple t_j from source d_i or not. Thereby, the cost of the purchase decision marked by DT is:

$$\sum_{i=1}^m PF[d_i](\{t_j | a_{ij} = 1, j = 1, \dots, n\}). \quad (4.3)$$

Let a sub-problem of PA be: find an allocation of solutions in S' and the purchase decision of its relevant tuples, DT , within the cost of b , where $S' \subseteq S$ and $b \leq B$. We

define the 3D-matrix answer of the sub-problem as $Tri - M[DT, S', b]$. It marks a set of the allocated solution, Λ , and the purchase decision for all the relevant tuples in DT .

Algorithm 2: 3DDP Allocation

Input: S , a set of solutions;
 ϵ , a small positive real number;
 B , budget constraint.
Output: Λ , an allocation of S .

```

1  $S', T \leftarrow \emptyset$ ;
2 foreach  $s$  in  $S$  do
3    $Tri - M(DT, S', 0) \leftarrow \emptyset$ ;
4    $b \leftarrow \epsilon$ ;
5   while  $b \leq B$  do
6      $z \leftarrow Zip(T \cup RT(s)) - Zip(T)$ ;
7      $DT' \leftarrow$  Decisions made by  $Zip(T \cup RT(s))$ ;
8      $Reject \leftarrow$  the utility of  $Tri - M(DT, S', b)$ ;
9      $Accept \leftarrow$  the utility of  $\{s\} \cup Tri - M(DT, S', b - z)$ ;
10     $S^{NEW} \leftarrow S' \cup \{s\}$ ;
11    if  $Reject > Accept$  then
12       $DT^{NEW} \leftarrow DT$ ;
13       $Tri - M(DT^{NEW}, S^{NEW}, b) \leftarrow Tri - M(DT, S', b)$ ;
14    else
15       $DT^{NEW} \leftarrow DT'$ ;
16       $Tri - M(DT^{NEW}, S^{NEW}, b) \leftarrow Tri - M(DT, S', b - z) \cup \{s\}$ ;
17    end
18     $T \leftarrow RTA(Tri - M(DT^{NEW}, S^{NEW}, b))$ ;
19     $S' \leftarrow S^{NEW}$ ;
20     $b \leftarrow b + \epsilon$ ;
21  end
22 end
23 return  $Tri - M(DT, S, B)$ ;

```

Considering that the greedy algorithm always allocates the local cheapest solutions and the high price of tuples set by ADPF (e.g. *FlatPF*), the data sources assigned with ADPF are at a disadvantage. To alleviate this, we adopt a reordering process into dynamic programming to generate allocations to privilege solutions which require more tuples from data sources with ADPF.

Algorithm 2 shows our 3DDP heuristic. It starts at the sub-problems with the empty set of candidate solutions, S' , and its relevant tuples, T (Line 1), and then iterates candidate solutions in S (Lines 2-22) with intermediate budgets adding from 0 to the budget constraint by ϵ (Lines 4-21). For each solution loop, first initiate the 3D-matrix answer to the sub-problem with a budget of 0 as an empty set, and the intermediate budget as ϵ (Lines 3-4). Next, while the intermediate budget is no greater than the budget constraint, calculate the utilities of the rejection and acceptance of allocating the solution,

respectively, (Lines 6-9) and generate the new candidate solution set by adding the solution into the previous one (Line 10). After that, compare the utilities and update the 3D-matrix answer to the sub-problem with the new candidate solution set and budget by the decision with the higher utility (Lines 11-17). By the end of a loop, recalculate the relevant tuple set of the current answer and candidate solution set, and increase the intermediate budget by ' ϵ '. (Lines 18-20). When the iteration finishes, it obtains and returns the 3D-matrix answer to the original question (Line 23).

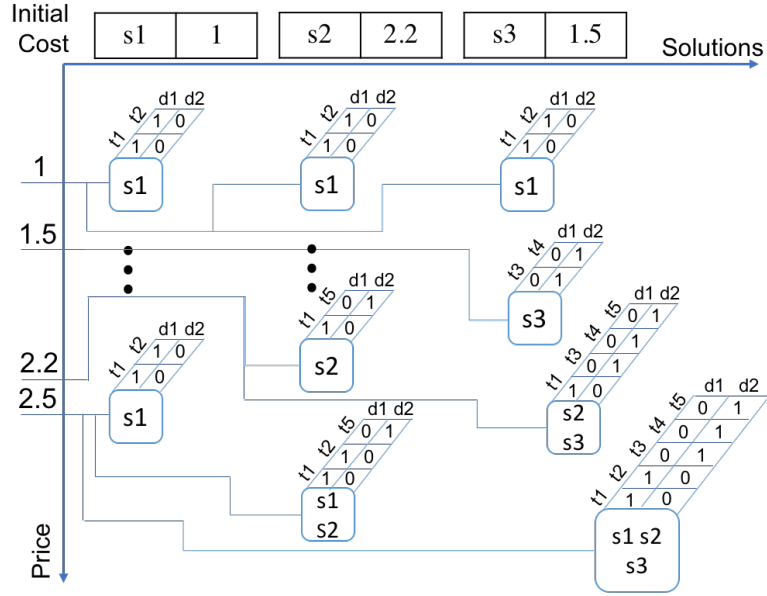


FIGURE 4.2: 3DDP Algorithm Solving the Example Query within a Budget of 2.5

In the 3DDP algorithm, it is challenging to choose an appropriate increase step (ϵ) of budgets. Yet that choice defines the sub-problems and impacts the performance of 3DDP. First, setting a large step may terminate the algorithm before the optimal answer is found. For instance, Figure 4.2 illustrates the process of 3DDP solving the example question. A budget step of 1 will lead to the termination of the algorithm at the sub-problem with a budget of 2 and miss the optimal answer at the cost of 2.5. Although a smaller step can avoid such an issue, neither does a small step perform the best. Considering the price of tuple sets, the step needs to be as small as the minimum price gap of tuple sets so as to cover all the possible sub-problems (0.3 in the example question). However, this step-size requires computing a large number of sub-problems. An appropriate budget step should, therefore, be neither too great to skip a possible sub-optimal answer nor too small to increase computation.

The search space of the 3DDP algorithm equals the number of sub-problems, which is $k \times B/\epsilon$. Thus, the complexity of the 3DDP algorithm is $O(n^2 \times k \times B/\epsilon)$.

TABLE 4.2: Settings of the Budget (B), Range of Flat-rate Price (P) and Free Quota (N) of Pricing Functions for Queries

Query	$ S $	B	Range of P	N
CD1	90	54.0	$[0, 60]$	30
CD6	11	6.60	$[0, 10]$	4
LD1	308	184.8	$[0, 200]$	102
LD2	185	111.00	$[0, 120]$	60
LD3	159	95.40	$[0, 100]$	53
LD4	50	30.00	$[0, 40]$	17
LD5	28	16.80	$[0, 20]$	9
LD6	39	23.40	$[0, 30]$	13
LD8	22	13.20	$[0, 20]$	7
LD11	376	225.6	$[0, 250]$	125
LS6	28	16.80	$[0, 20]$	9
LS7	144	65.40	$[0, 70]$	48

4.4 Evaluation

The goal of our evaluation is to investigate the performance of our two heuristic algorithms and characterise the impact of different pricing functions and valuated utilities of solutions. Before we conduct our experiments, we use the FedMark (Grubenmann et al., 2018a) query engine to collect the complete solutions to the existing widely-used queries from FedBench benchmark (Schmidt et al., 2011) and the relevant triple set of each solution of each query. Schmidt et al. (2011) described FedBench as ‘a comprehensive benchmark suite for testing and analyzing both the efficiency and effectiveness of federated query processing on semantic data’. There are 25 queries and 9 related data sources covering the field of Cross Domain Queries (CD), Life Science Queries (LS) and Linked Data (LD). As in the work of FedMark, we exclude queries from FedBench with fewer than 3 solutions, and those with solutions that have fewer than 3 relevant tuples, because it is very easy to find their optimal allocations. Table 4.2 shows the size of S for each query.

With the collected complete solution sets to the listed queries and their relevant triple sets, we implement our experiments in three steps. First, we reorganise the data sources from FedBench. The 9 different data sources from FedBench have no overlap. To create overlap and reduce the complexity of setting up various pricing functions for 9 data sources, as explained later in Chapter 4.4.1, we mesh all the tuples from the 9 data sources together, and then randomly copy them into 2 new data sources. Then, based on the new data sources, we map and record the source of each tuple that is relevant to any solutions. Next, we set the budget constraints for each query and execute our purchase allocation algorithms.

Building on that, we vary the types and parameters of the pricing functions to demonstrate the performance of the algorithms and answer two questions: Does the heuristic algorithms' performance change depending on the type and parameters of ADPF? How do different utility distributions of solutions affect the performance of the heuristics? Our evaluation is motivated by the practical question: Does one heuristic always dominate the other? Or can the market decide which one to use based on input parameters?

4.4.1 Experiment Settings

Data Sources Preparation When each of the tuples relevant to a set of solutions is available from one, and only one, data source, there is no decision to make about selecting where to buy from. As FedBench sources have no overlap, we generate two overlapping data sources, d_1 and d_2 , based on FedBench, as follows: first, we mesh and copy all the tuples in the nine original data sources from FedBench into a big dataset and create a numeral index for each tuple; second, according to the index, we copy every 10,000 tuples alternatively to d_1 and d_2 ; last, to generate overlaps, we randomly duplicate 40% of every 100 tuples to d_1 and the other 60% to d_2 . This guarantees that at least 50% of the data overlaps between d_1 and d_2 . Although we only create 2 data sources to reduce the complexity of varying the parameters explained later in this chapter, the capability of our algorithms is not limited by the settings of this experiment and we plan to extend the size of data sources and the variety of duplication rate, queries, pricing functions and budgets in our future work.

Pricing Functions and Parameters We assign d_1 a fixed price function with prices chosen uniformly at random with a value in $[0, 1]$, notated as $UBPF$. We assign two different functions to d_2 : $FlatPF[P]$ and $FMPF[P, N]$, yielding two experimental configurations that we call $UBPF + FlatPF$ and $UBPF + FMPF$. We choose $FlatPF$ and $FMPF$ since, according to [Liang et al. \(2018\)](#), they are the most popular ADPFs for digital content. The settings of all parameters in pricing functions are listed in Table 4.2, and are calculated as follows:

- Flat Price (P): ranges from 0 to the next multiple of 10 greater than the budget of a query in both pricing functions;
- Free Quota (N): one third of $|S|$ of a query. Note that a large free quota would allow an easy allocation of all the relevant tuples, so we choose to fix a lower quota for our experiments.

Note that we only use the three types of pricing functions listed in Table 3.1 and skip the free pricing function. The reasons are that the above setting covers the case that d_2 price all their data at a cost of 0 (when $P = 0$), and the free pricing function has

TABLE 4.3: Sized-Distributions of Utilities

Sized-Distribution	# of S	$v(s)$
Small Difference	25%	1.25
	75%	1
Medium Difference	25%	1.5
	25%	1.25
	50%	1
Large Difference	25%	1.75
	25%	1.5
	25%	1.25
	25%	1

minimum impact on the problem of purchase allocation within a limited budget. In future work, we intend to extend the number of data sources and include various pricing functions so as to generate different combinations, i.e. different data sources using different pricing functions can sell duplicated data.

Budget The setting for budgets in our experiment aims to simulate enquiries that buyers can not afford a complete set query answer but only a subset of it, i.e. an allocation. Considering that a solution requires at least 3 tuples which are priced uniformly from 0 to 1 by *UBPF* (we do not consider *FMPF* and *FlatPF*, since we vary their parameters in the experiments), we assume that the cost of a solution is around 1, and set the budget to 60% of the number of solutions in the complete query answer to a query, as detailed in Table 4.2.

Utility We create two experimental configurations of solution utilities: equal-utility and sized-distribution. *Equal-utility*, where all the solutions have the same utility value of one. Fixing the utility variable allows us to examine the impact of pricing functions on the algorithms' performance. *Sized-distribution*, where we assign different utility values to a varying percentage of solutions to simulate different utility distributions and study the impact of utilities on the approach. We design 3 sized-distributions of solution utilities: (1) Small Differences (SD) increase the utility value of 25% solutions from 1 to 1.25; (2) Medium Differences (MD) raises the utility value of 50% solutions by increasing another 25% utility value from 1 to 1.5 based on SD; and (3) Large Differences (LD) increases the utility value of 75% solutions by increasing another 25% utility value from 1 to 1.75 based on MD.

Hardware and Metrics We conduct our experiments on a Red Hat Enterprise Linux 7 (RHEL7) virtual machine with 4 CPU Cores, 32GiB RAM and 200GiB storage, and then we evaluate the resulting allocations according to three metrics:

TABLE 4.4: Comparison Metrics for Size

Abbr.	Description
#U	# of P s.t. $ \Lambda_p^D = \Lambda_p^G $
#W	# of $P < B$ s.t. $ \Lambda_p^D > \Lambda_p^G $
#F	# of $P < B$ s.t. $ \Lambda_p^D < \Lambda_p^G $
#H	# of $P > B$ s.t. $ \Lambda_p^D > \Lambda_p^G $
#G	# of $P > B$ s.t. $ \Lambda_p^D < \Lambda_p^G $
DA	$avg(\Lambda_p^D - \Lambda_p^G)$

- *Utility*: the total utility value of solutions in an allocation. Under the Equal-utility setting of solution utilities, we compare the size of the solutions in allocations.
- *Price Rate*: the average price of solutions in the returned allocation. If the size of the allocation is equal, a lower price rate means that one algorithm was more budget-efficient than the other.
- *Runtime*: time for the algorithms to produce allocations.

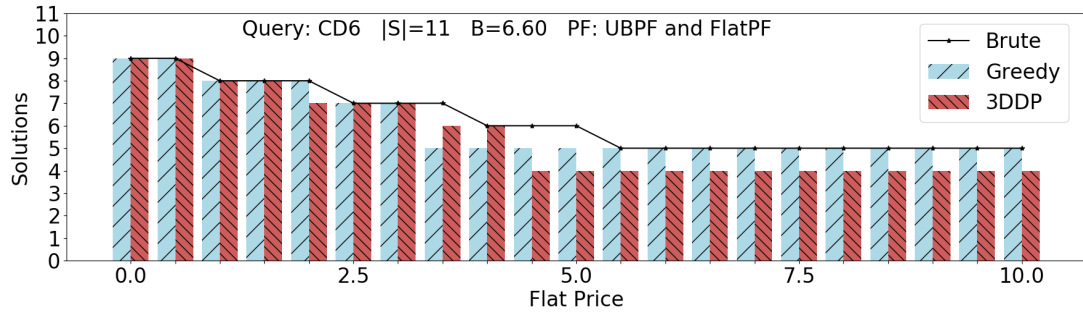
To sum up the size comparison of 3DDP and Greedy over the price range, Table 4.4 defines the six sub-metrics where Λ_p^D and Λ_p^G represent, respectively, the allocations returned by 3DDP and Greedy at a price point p .

- #U, the amount of price points where 3DDP and Greedy return the same size allocations;
- #W and #F, the number of price points below the budget where 3DDP allocates more and less solutions than Greedy, respectively;
- #H and #G, the number of price points over the budget where 3DDP allocates more and less solutions than Greedy, respectively;
- DA, the average difference between the size of Λ_p^D and Λ_p^G .

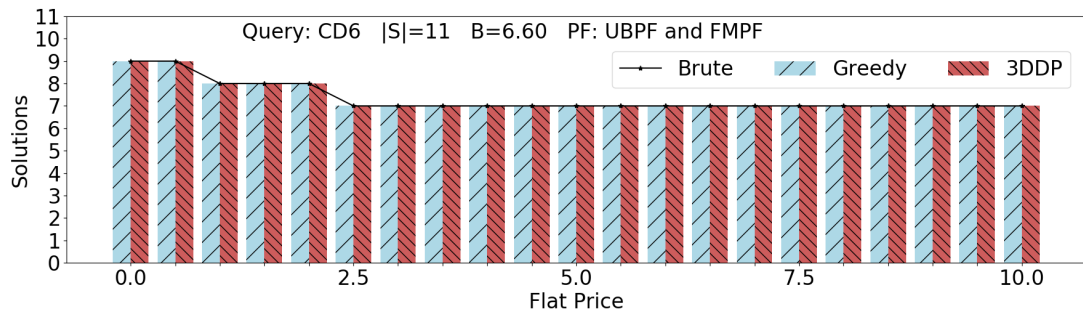
Intuitively, a larger #U suggests that Greedy and 3DDP are equally effective. #W greater than #F indicates 3DDP performs better when the flat-rate prices are less than the budgets. Otherwise, Greedy is better. Similarly, #H greater than #G indicates 3DDP performs better when the flat-rate prices are greater than the budgets. Otherwise, Greedy is better. DA calculates the difference in size between the allocations from 3DDP and Greedy on average.

4.4.2 Impact of Pricing Parameters

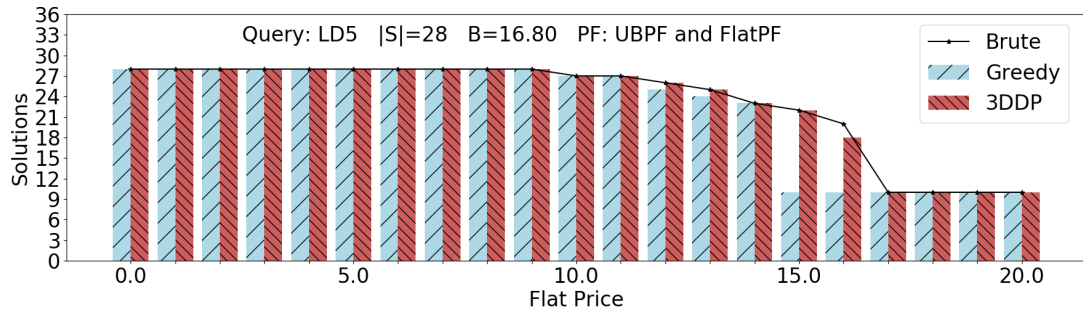
To measure how far Greedy and 3DDP are from the optimal answers, we implement a brute-force algorithm as the baseline for performance. We run the brute-force algorithm with a timeout of 24 hours. Unfortunately (but expectedly), the brute-force algorithm times out or overflows memory in all queries but: CD6, LD5, LD8 and LS6. Figure 4.3



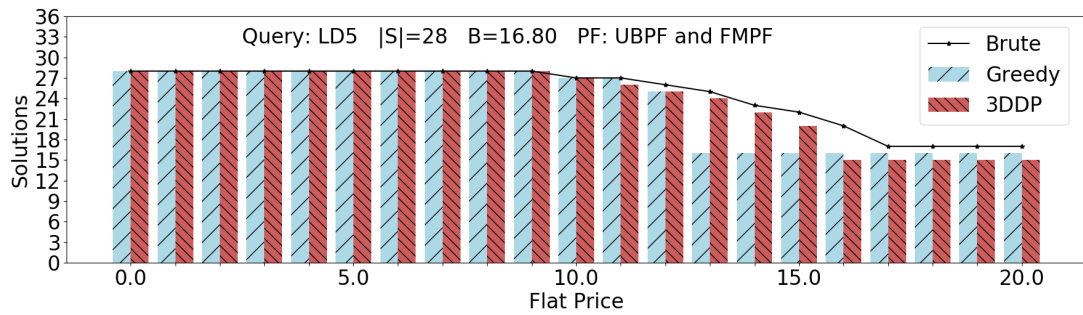
(a) CD6 with UBPF + FlatPF



(b) CD6 with UBPF + FMPF



(c) LD5 with UBPF + FlatPF



(d) LD5 with UBPF + FMPF

FIGURE 4.3: Total Solutions in the Returned Allocations and Runtime of Brute Solution, Greedy and 3DDP (1)

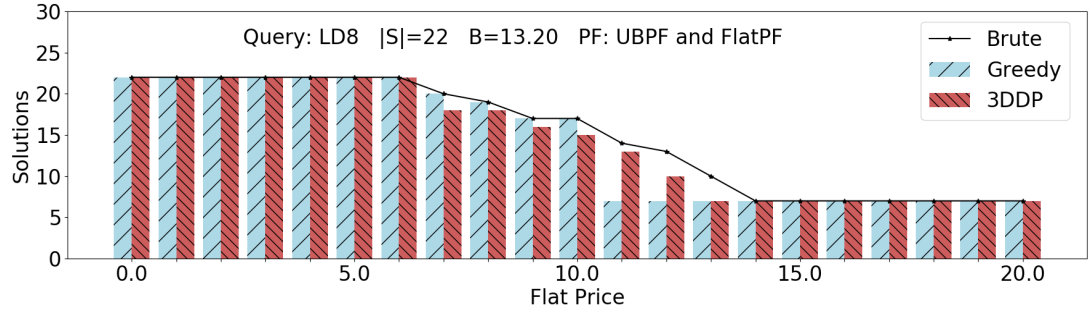
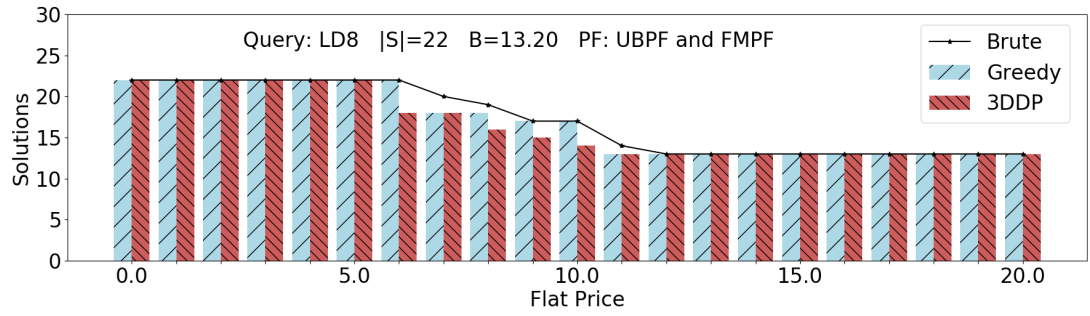
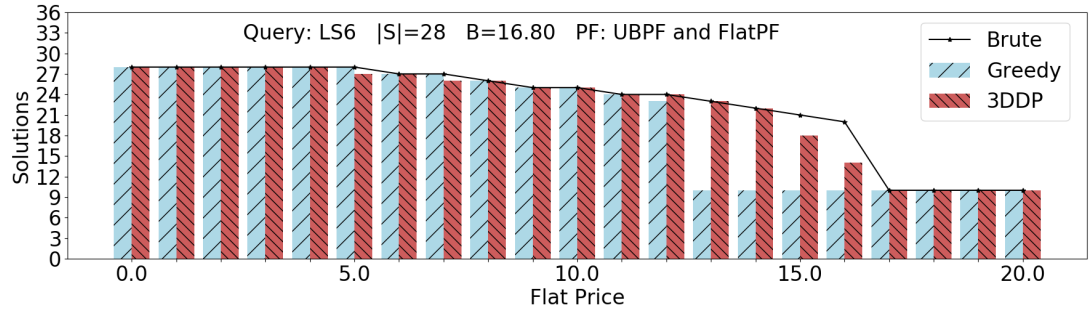
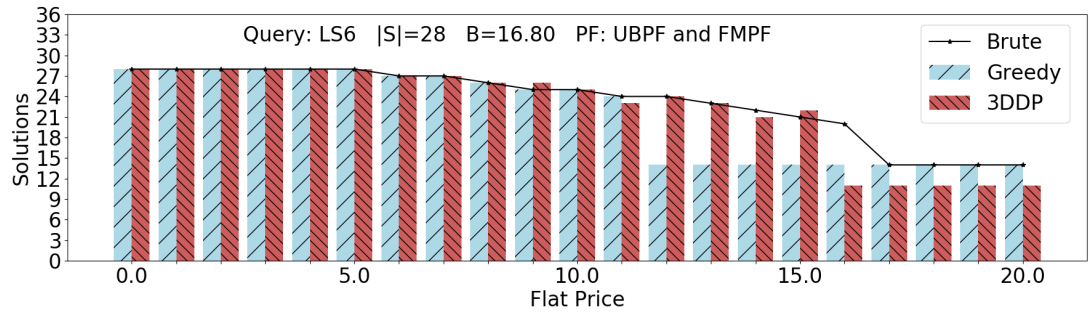
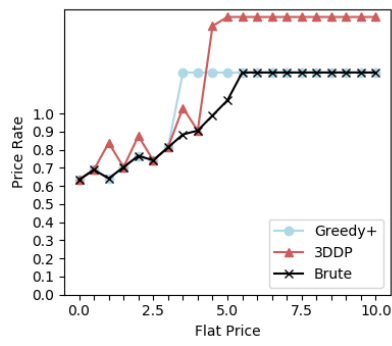
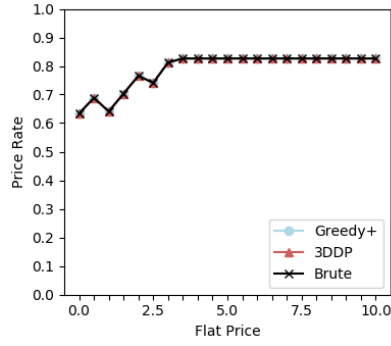
(a) LD8 with *UBPF* + *FlatPF*(b) LD8 with *UBPF* + *FMPF*(c) LS6 with *UBPF* + *FlatPF*(d) LS6 with *UBPF* + *FMPF*

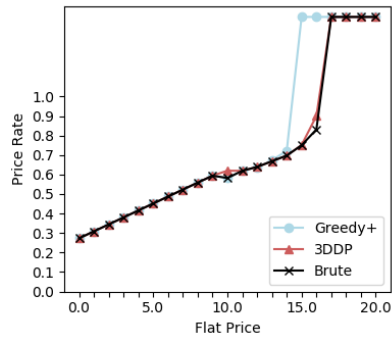
FIGURE 4.4: Total Solutions in the Returned Allocations and Runtime of Brute Solution, Greedy and 3DDP (2)



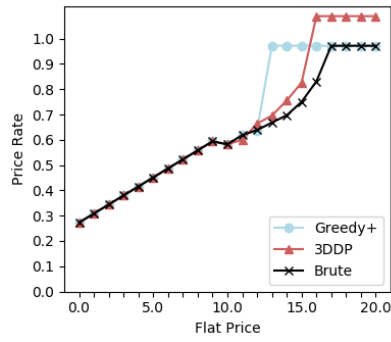
(a) CD6 with UBPF + FlatPF



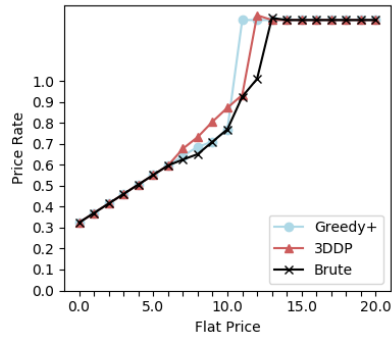
(b) CD6 with UBPF + FMPF



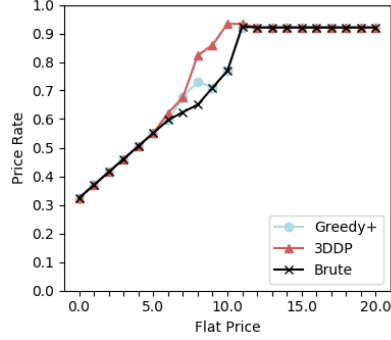
(c) LD5 with UBPF + FlatPF



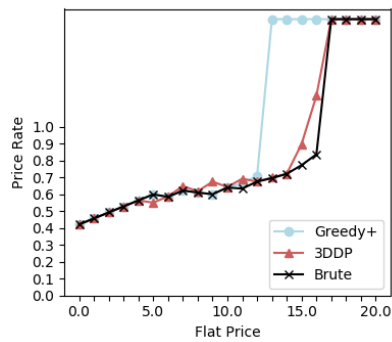
(d) LD5 with UBPF + FMPF



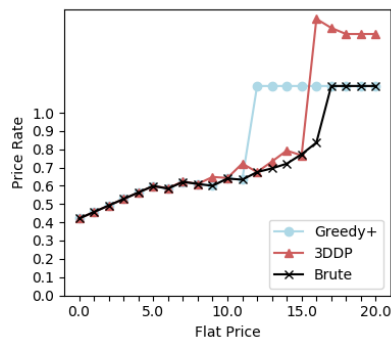
(e) LD8 with UBPF + FlatPF



(f) LD8 with UBPF + FMPF



(g) LS6 with UBPF + FlatPF



(h) LS6 with UBPF + FMPF

FIGURE 4.5: (continued) The Price Rate of Allocations Generated for Query

- 4.5 compare the results of Greedy and 3DDP with respect to the brute-force baseline for the configurations of $UBPF + FlatPF$ and $UBPF + FMPF$ in terms of size and price rate metrics, respectively.

In Figure 4.3 and 4.4, 85.71% allocations either returned by Greedy or 3DDP are optimal. The overall non-optimal allocations returned by Greedy have 26.62% fewer solutions than the optimal on average, while 3DDP has 14.83%. The price rates of 3DDP's allocations stay very close to the optimal, as Figure 4.5 shows. We observe a decrease in the number of solutions found by 3DDP when the budget is less than the price of whole solutions. The reordering process in 3DDP is the main reason causing this. For instance, all the solutions for Query CD6 ask relevant tuples from d_1 but the reordering process leads 3DDP to purchase from d_2 , thus, 3DDP misses the optimal answers when the flat price of d_2 exceeds the budget.

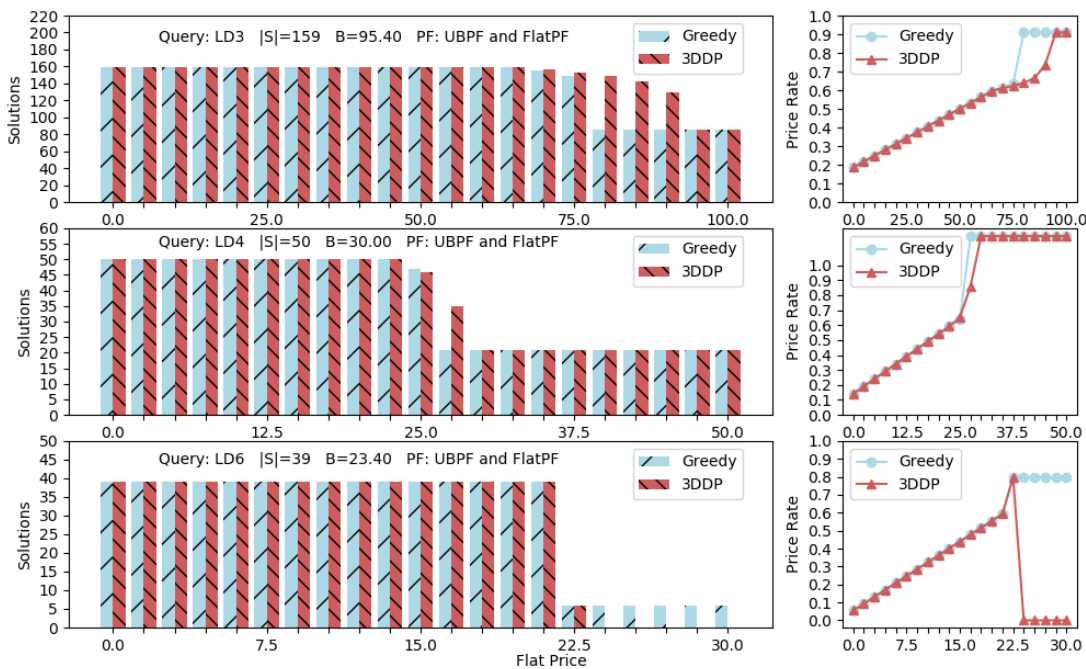
TABLE 4.5: The Difference Comparison of Λ_P^D and Λ_P^G over Flat Price P and Runtime

(a) UBPF+FlatPF								(b) UBPF+FMPF							
	S	#U	#W	#F	#H	#G	DA	#U	#W	#F	#H	#G	DA		
CD1	90	21	0	0	0	0	0	21	0	0	0	0	0		
CD6	11	6	2	5	0	8	-0.73	21	0	0	0	0	0		
LD1	308	16	3	0	2	0	77.4	14	5	0	0	2	26.14		
LD2	185	18	0	0	3	0	46.67	16	0	0	1	4	-0.8		
LD3	159	16	2	0	3	0	34.2	14	3	2	0	2	-3.43		
LD4	50	19	0	1	1	0	6.5	10	1	1	0	9	-7.09		
LD5	28	17	2	0	2	0	5.5	12	3	1	0	5	1.33		
LD6	39	16	0	0	0	5	-6.0	21	0	0	0	0	0		
LD8	22	15	0	3	2	1	0.5	17	0	4	0	0	-2.75		
LD11	376	13	6	1	1	0	123.88	12	6	0	0	3	72.11		
LS6	28	14	5	2	0	0	5.14	10	5	2	0	4	1.73		
LS7	144	10	10	0	1	0	27.73	9	6	4	0	2	6.83		
Sum	756	181	30	12	15	14	320.79	177	29	14	1	31	94.07		

Table 4.5(a) shows the comparison between the allocations of 3DDP and Greedy on the experiment in the configuration of $UBPF + FlatPF$. 3DDP performs better by allocating more solutions than Greedy for 9 out of 12 queries on average. Most of these queries are the ones with the largest solution size. 3DDP shows its allocation advantage on average over 25% of the size of the answer for the queries. In one of them (LS7), according to the definition of #U and #W, 3DDP keeps its advantage on more than 50% of the price points, and performs equally with Greedy at the others. In 2 queries (LD4, LD8), 3DDP performs better at 1 and 3 price points according to #F. For one query (CD1), both heuristics returned exactly the same results for all the price points. Moreover, for three queries (CD6, LD6, LS6), Greedy stands out at 13, 5 and 2 price points that are greater than the budgets, according to #G.

TABLE 4.6: Runtime of Experiment $UBPF + FlatPF$ and $UBPF + FMPF$

(a) Runtime(s) of $UBPF + FlatPF$			(b) Runtime(s) of $UBPF + FMPF$	
	Greedy	3DDP	Greedy	3DDP
CD1	0.149	28.227	0.162	2.533
CD6	0.006	0.039	0.006	0.067
LD1	35.082	5337.137	33.354	1440.672
LD2	3.081	637.945	3.302	95.636
LD3	2.245	315.03	2.024	92.485
LD4	0.102	2.589	0.107	2.854
LD5	0.03	0.417	0.029	0.528
LD6	0.024	0.491	0.029	0.402
LD8	0.016	0.155	0.016	0.314
LD11	175.297	18955.612	181.908	8137.892
LS6	0.053	0.81	0.042	0.969
LS7	2.214	109.502	1.899	104.761
Average	18.192	2115.663	18.573	823.258

FIGURE 4.6: Total Solutions And Price Rate of the Allocation of Greedy And 3DDP for Query LD5, LD6, LD8 over FlatPF[P] with Different P

To illustrate the trend of size and price rate metrics, Figure 4.6 shows the details of Query LD3, LD4 and LD6. Note that we only select and show the results of Query LD5, LD6 and LD8 to explain the common wedge of the results of all the queries. For the complete results charts, please refer to Appendix A.2. We observe a common three-state-pattern as the flat price grows. In the first state, both heuristics perform equally at low price points and on the same price rate; then after a certain price point, the advantage of 3DDP shows up by allocating an average 32.22% more solutions for Query

LD3 and LD4, while maintaining lower price rates; finally, around the point where the flat price becomes larger than the budget, 3DDP loses most of its advantage, and even performs worse than Greedy in some cases (Like LD6). The advantage of 3DDP and its decline with respect to Greedy, are mainly created by the reordering process. When $P < B$, it helps 3DDP purchase more tuples from d_2 who uses FlatPF to allocate more solutions and reduce the price, while Greedy allocates the local cheapest solution without considering the pricing functions of data sources.

Table 4.5(b) compares of Greedy and 3DDP in the $UBPF + FlatPF$ configuration experiment. In contrast with the previous results, 3DDP shows less of an advantage over Greedy. Both heuristics return the same number of solutions at all price points in 3 queries. 3DDP performs better at most price points for Query LD1, LD11 and LS7. Interestingly, Greedy is slightly better in Query LD3 and LD4, where 3DDP has the largest advantage in the $UBPF + FlatPF$ experiment.

Figure 4.7 shows detailed results for 3 queries. We notice a very similar three-state-pattern to the $UBPF + FlatPF$ configuration experiment: as the price grows, both heuristics are equally good for lower prices, and then at a certain price point, 3DDP starts to return better results, up to the point where $P > B$; after that point, 3DDP retains a small advantage or Greedy is slightly better. Interestingly, the left group of sub figures in Figure 4.7 shows that Greedy gains a lower price rate than 3DDP at over half price points, and even the entire price range for Query LS7. This suggests that a free quota of data favours the Greedy approach in allocating solutions and efficiently spending the budget.

In conclusion, 3DDP has an advantage over Greedy for purchase allocation when $P < B$, especially for FlatPF, but it has a much lesser advantage when $P > B$, especially for FMPF.

Runtime: Since dynamic programming needs to calculate every sub problem and store the results, we do not expect 3DDP to be faster than Greedy. Table 4.6(a) and 4.6(b) lists the average runtime of Greedy and 3DDP in both experiments. The worst case for 3DDP is around 200 times higher than Greedy for Query LD2. In the best case, 3DDP runs 7 times longer for Query CD6.

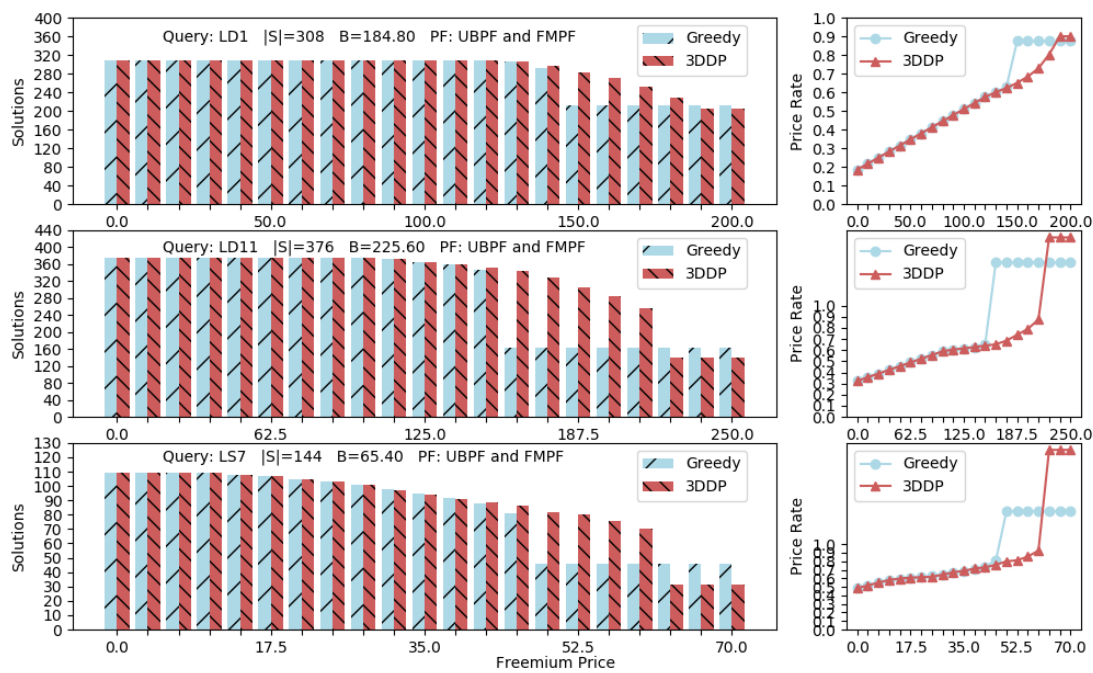


FIGURE 4.7: Total Solutions And Price Rate of the Allocations of Greedy And 3DDP for Query LD5, LS6, LS7 over FMPF[P, N] with Different P

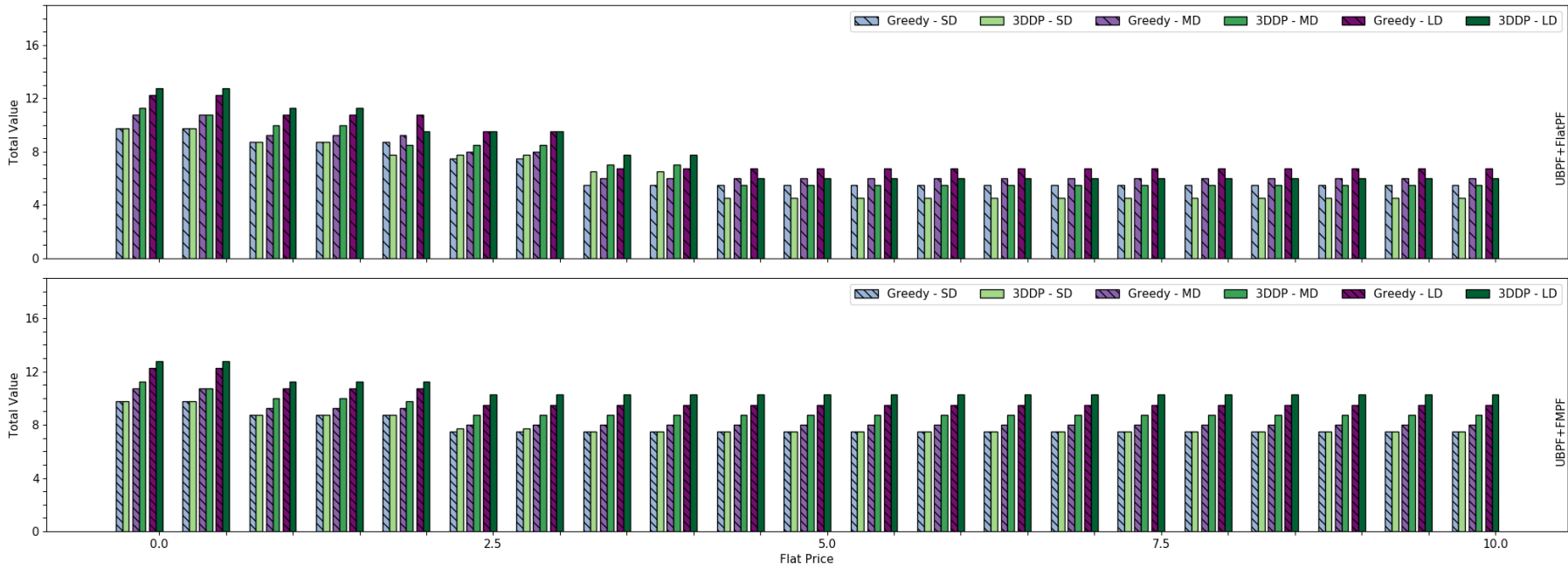


FIGURE 4.8: Total Utility of the Allocations Returned by Greedy and 3DDP for Query CD6 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities

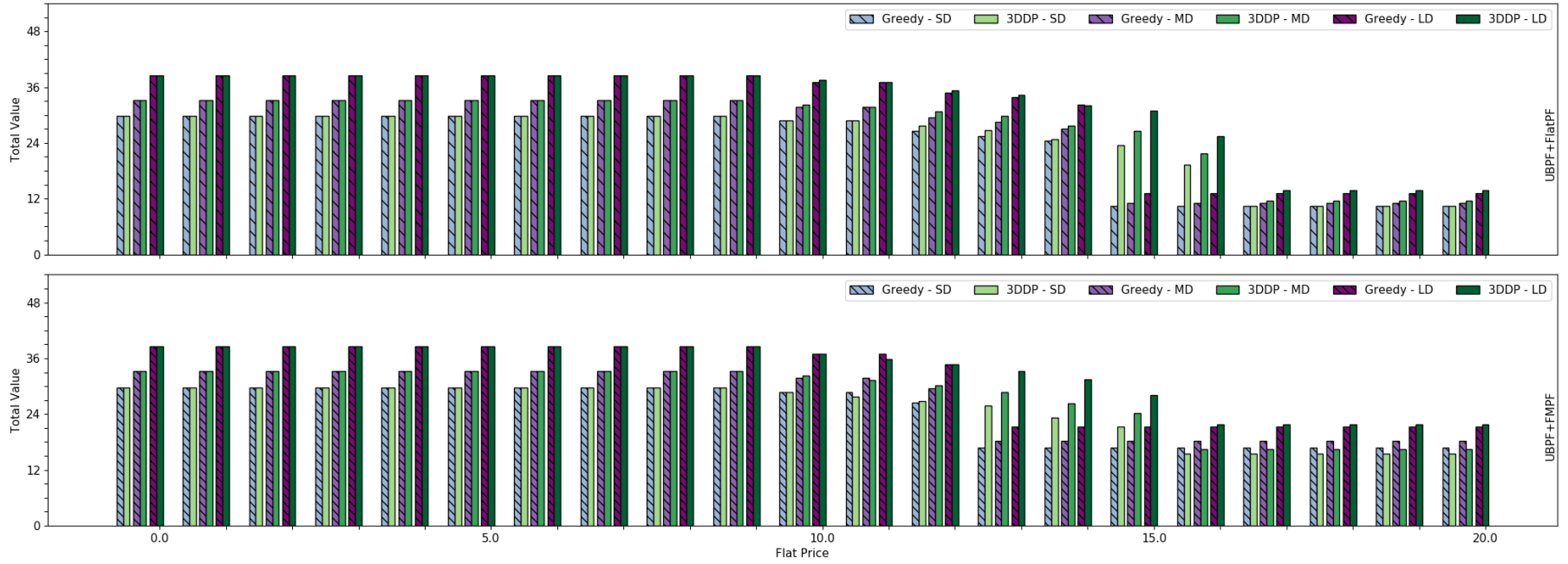


FIGURE 4.9: Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD5 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities

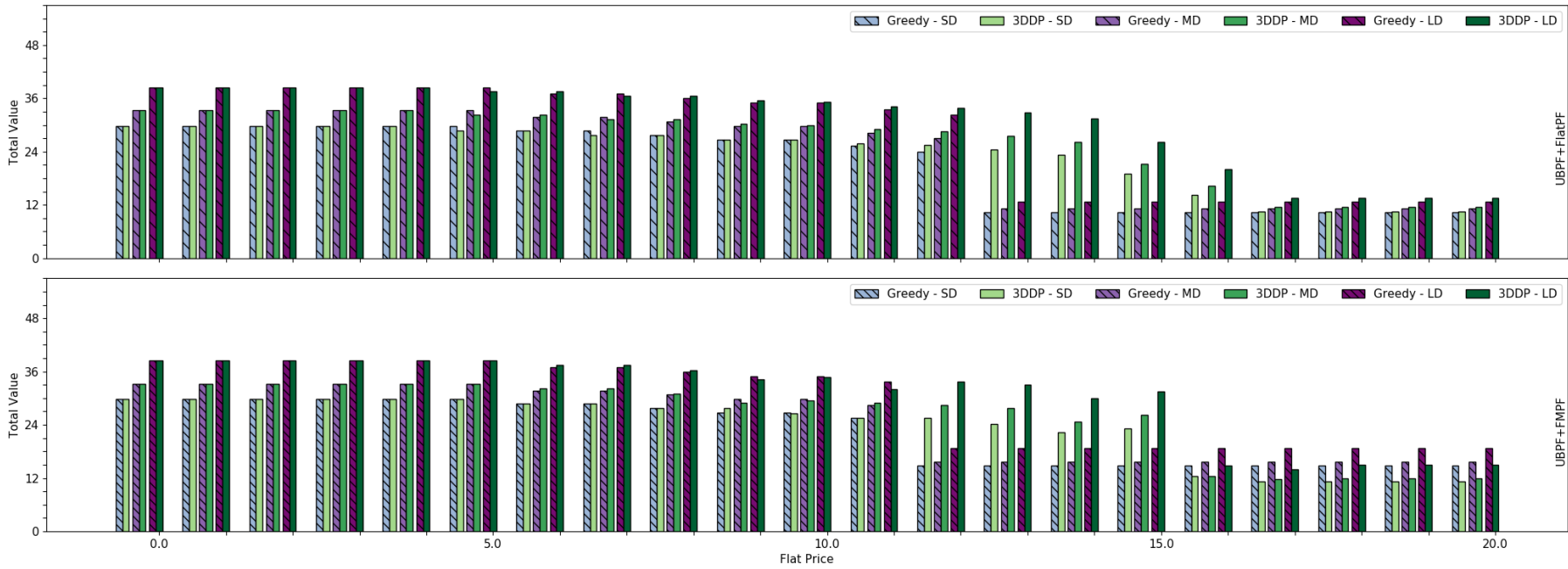


FIGURE 4.10: Total Utility of the Allocations Returned by Greedy and 3DDP for Query LS6 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities

4.4.3 Impact of Utility Distribution

We consider three different utility sized-distributions: SD, MD and LD, detailed in Table 4.3, as a simulation of utility valuations. We explore the interplay between optimising the utility of returned allocations with pricing functions in configurations of $UBPF + FlatPF$ and $UBPF + FMPPF$.

In Figure 4.8-4.10, we show the results for Query CD6, LD5 and LS6 and present the total utilities of allocations returned by Greedy and 3DDP when the utility of solutions increases from case SD to LD with configurations of $UBPF + FlatPF$ and $UBPF + FMPPF$ in two rows, respectively, to compare against our initial exploratory experiment. For completeness, we have the figures for all the queries in Appendix A.3.

Compared to the three-state-pattern observed before, the performance comparison between 3DDP and Greedy shows complexity: at price points lower than the budget, the utility of allocations returned by 3DDP is higher than Greedy in both experiments, except one price point for each query; Greedy still performs better when P is far more greater than B in $UBPF + FlatPF$ experiment of Query CD6 and $UBPF + FMPPF$ experiment of LS6, but 3DDP spreads its advantage to 4 more settings of P for Query CD6, 2 for LD5 and 9 for LS6 in $UBPF + FlatPF$ experiments, compared to Figure 4.3 and 4.4; in $UBPF + FMPPF$ experiment, 3DDP holds allocations with higher utilities even at the price points greater than the budget, both in case MD and LD, except for Query LS6, which is the same for equal utility settings. To summarise, when the utility difference is much larger (MD and LD), 3DDP gains higher utility for its allocations than Greedy, especially in $UBPF + FMPPF$ experiment. For the slight variation of the utility distribution of solutions (SD), the performances of 3DDP and Greedy are similar to the configuration of Equal-utility.

In conclusion, the performance of 3DDP and Greedy for solving the PA problem is affected by both pricing functions and utility distributions of solutions. When a data source applies $UBPF + FlatPF$ and buyers have budgets higher than its flat price, 3DDP helps the data marketplace to get a better allocation with higher utility and more solutions. If the budget of a buyer is lower than the flat price, however, the marketplace should use greedy over 3DDP. When the data source turns to $UBPF + FMPPF$, we suggest the marketplace use Greedy when buyers utilise all solutions equally. On the other hand, with a variable utility valuation, our results suggest that 3DDP is a better option to gain a higher utility allocation.

4.5 Summary

In this chapter, we have described a federated data marketplace, Data Emporium, which enables sellers to price their data autonomously and enables query answering based on

input budget constraints and utility valuations from buyers. We define a general form of this problem as the *Purchase Allocation Problem* (PA). Previous federated marketplace models that solve the PA problem are limited to predefined data tuple prices, but a large number of pricing strategies currently used in practice are access-dependent pricing functions (ADPF), e.g. flat-rate and freemium access.

We extend previous models to solve the PA problem in the presence of ADPF. First, we identify the problem of minimising the cost of a set of solutions across multiple sellers that may offer the same tuples with different prices, and we propose the cost compression algorithm to solve it. We use the cost compression algorithm to develop two heuristic solutions and compare how their execution time and the size, utility and price rate of their allocations change with variations of pricing functions and solution utilities. Our results suggest that Greedy is much faster, but 3DDP finds better allocations, especially when the flat price P of the tested pricing functions is less than the budget. However, when P is greater than the budget, the advantage decreases, suggesting that Greedy should be used in that case to take advantage of its faster execution time. When the variability in the utility value of solutions increases, 3DDP performs better than Greedy.

Chapter 5

Free Market

The motivation for the design of a new data market comprises the following considerations: (1) when sellers doubt the equivalence of their incomes and contributions, they may stop offering free access to the marketplace or quit, leaving the marketplace or collaborators no choice but to face the risk of losing business; (2) when buyers lose their interest in being victims to high bills and query answers that are not customised to their constraints, both sellers and marketplaces face the risk of losing business; (3) when a single seller or a marketplace fails to answer a query, buyers face the challenge of purchasing data from multiple independent sellers or competing marketplaces. Hence, we propose a Free Market, which lets buyers and sellers exchange or share data directly. This implies that Free Market cuts out the third party and does not require the trust of sellers and buyers which is used to be asked by a marketplace. Figure 5.1 presents the architecture of the Free Market. The Free Market also consists of three types of participants: buyers, sellers and a marketplace as existing markets. Differently, it will leave the progress of query processing to sellers and the join process to buyers. As Figure 5.1 illustrates, regardless of locations and scales, sellers autonomously manage their data sources in terms of data access control, processing queries and charging the results of them, i.e. pricing query answers and receiving payments directly from buyers. Instead of being responsible for managing the data sources of sellers and searching for answers to the requirements of buyers, the marketplace in the Free Market only takes charge of publishing a catalogue of sellers in the market and updating the status of sellers in that catalogue when sellers come and go with the data sources. Buyers utilise a local query engine to collect information in the published catalogue and search for answers to their requirements.

In this chapter, we will first explain the design of the catalogue in the Free Market, and then introduce the local query engine that supports buyers to purchase data in the Free Market.

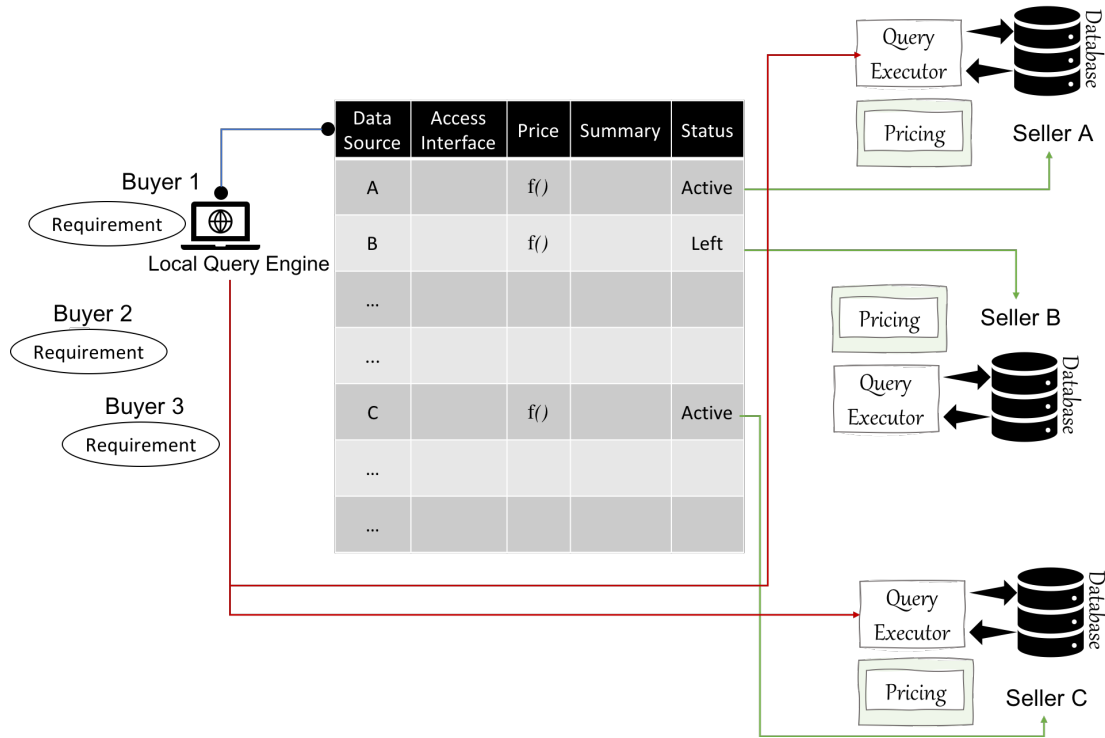


FIGURE 5.1: Architecture of the Free Market (The green lines represent the corresponding relationship between the records in the catalogue and the sellers in the market. The red lines illustrate the conducted access or data requirements from buyers to the selected sellers.)

5.1 Market Catalogue

The central table in Figure 5.1 presents our design of the catalogue, which is open and free for buyers. The information about an available data source in the catalogue consists of its access interface, pricing strategy, summary statistics and status.

5.1.1 Summary statistics of Data Sources

As [Grubenmann et al. \(2017a\)](#) and [Heling and Acosta \(2020\)](#) claimed, without free access to data sources or statistical information, it is impossible to gain relevant knowledge for a query, let alone optimise it. This means restricted settings, such as a lack of open information about data sources, are an obstacle for selling data to buyers when they have obvious difficulty querying and paying. Considering the goal of monetising data, the Free Market supposes that each seller who locates a data source in the Free Market is willing to advertise information about the data source. In this work, we define the advertising information of a data source as summary statistics that contain the metadata and statistics of the data source. Hence, these summary statistics allow sellers to attract buyers, while also allowing buyers to conduct their data searching with sufficient information about a data source.

TABLE 5.1: Example RDF Dataset

Index	Tuple
1	student:Miya rdf:studyAt university:SOTON
2	student:Lily rdf:studyAt university:SOTON
3	student:Miya rdf:majorIn field:DataMarket

TABLE 5.2: Summary Statistics of Example Dataset

Predicates	Summary
rdf:studyAt	{ frequency: 2 subject: 2 object: 1 }
rdf:majorIn	{ frequency: 1 subject: 1 object: 1 }

To keep the summary statistics consistent for buyers to compare and select data sources, we assume that the Free Market defines or picks a specific pattern or format of the summary statistics for all data sources. We assume that a pattern of the summary statistics should consist of the frequency of keys in a data source that can be queried and, therefore, users can estimate the number of results before they send out enquiries. For instance, the pattern of statistics of a RDF dataset needs to list all the predicates along with its frequency, i.e. the number of different subjects and objects associated with a predicate. To be specific, for the dataset in Table 5.1, the summary statistics of it should at least include the information in Table 5.2. Therefore, for a given query, '*SELECT * WHERE ?subject rdf:studyAt ?object.*', the estimated number of results is 2. For other kinds of queries, we will explain our detailed estimation approach in Chapter 6.

5.1.2 Access Interface

The interface of data sources is a public gateway for buyers to access the data. There are multiple types of interfaces of data sources, e.g. the widely-used Virtuoso SPARQL Query Editor of DBpedia ([dbp](#)) and triple pattern-based query interface ([Verborgh et al., 2016](#)), the Application Programming Interface (API) which accepts programs to run, as mentioned in Chapter 2.

5.1.3 Pricing Strategy

Sellers publish pricing functions to declare how they will charge data requests, and to protect their profits, they ask for immediate payments when they return the required data to buyers. It is fair that sellers do not offer the required data to buyers before payments, since once data is seen, it is accessed. On the other hand, sellers need to guarantee that they will not charge buyers more than once for the same required data. The feasible method for this is a refund policy (Upadhyaya et al., 2016) or enabling data cache (Chidlovskii and Borghoff, 2000).

5.2 Local Query Engine

In the data mechanism described above, buyers search data sources for answers to their requirements based on the summaries published in the catalogue. This indicates that buyers are responsible for arranging query plans to access remote data sources. Therefore, it is crucial to have a local query engine able to optimise queries for buyers and then query available data sources in the market catalogue. Different from remote query engines which are located in a marketplace and have access to the data sources of all sellers, a local query engine works at the end of buyers to optimise the queries of buyers in light of the constraints, such as budgets and preferences/priorities, along with the queries, before then paying for each access of data sources.

The workflow of a local query engine is detailed in Figure 5.2. The initial setting with an incoming data requirement is a list of data sources. Considering the massive number of data sources over the Web, it is natural that buyers would set a list of reliable data sources to query instead of broadcasting queries all over the Web before query execution. The list is completely the buyers' choice and they can make it based on the announcement from the marketplace. Thus, a list of access interfaces of the preferred data sources is the first step to start a local query engine. The query optimisation in the local query engine works in two phases: source selection and query planning. The first phase selects data sources which include data that matches the query in a data requirement of a buyer. The selection is based on the summaries of the listed data sources in the catalogue of the marketplace. The second phase formulates a query plan which indicates the order in which sub-queries are sent out sub-queries to the selected data sources. The order is called a query plan and will be executed by a query engine. We assume that a query plan is a left-deep plan, based on the proof that any query plan can be transformed into a left-deep plan that costs no greater than the original query plan (Li et al., 2015). The execution of a query plan in a local query engine is slightly different from remote query engines located in marketplaces. Considering that buyers will pay the cost of intermediate query results at the time they receive them from each data source in the data mechanism of the Free Market, it would be natural that a local query

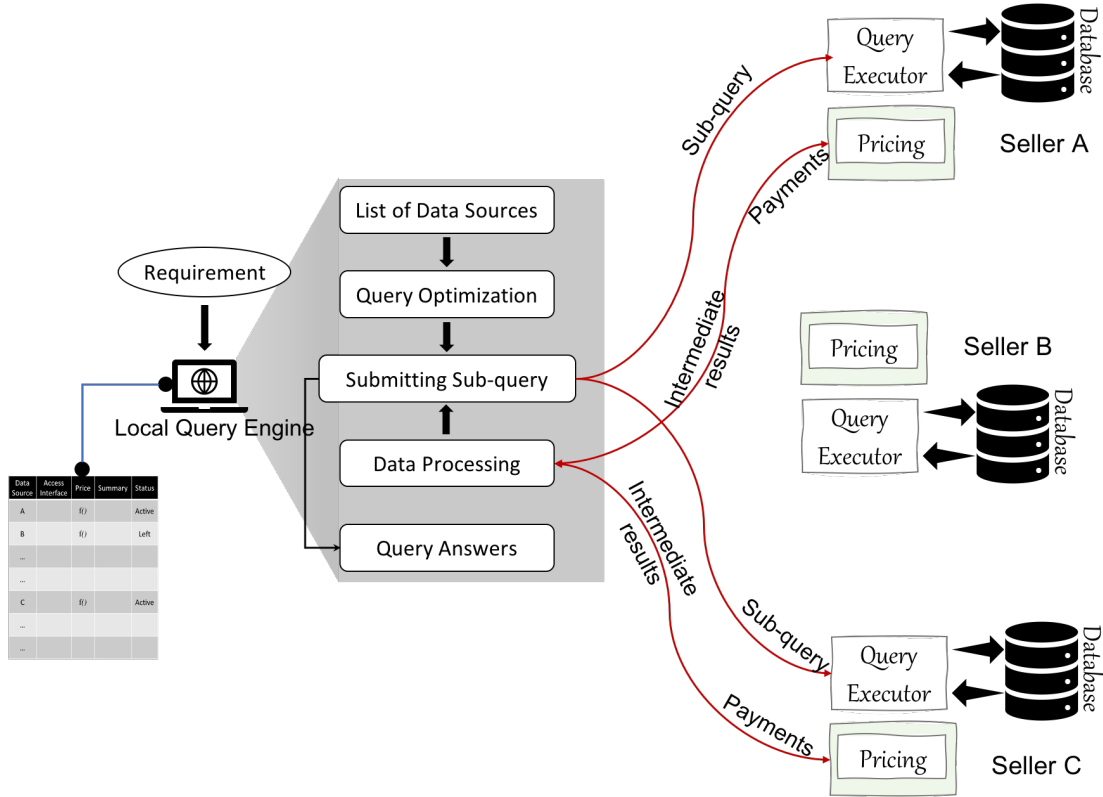


FIGURE 5.2: Workflow of Local Query Engine

engine could use previous results that they had already paid for in order to plan the following query requests during execution. Thereby, the execution following a query plan will send a sub-query to its selected sources, collect the returned intermediate results but join these results to the next sub-query before sending it out, until the final query answers are attained. Such a local join process can reduce the number of intermediate results that need to be collected and, thus, reduce the cost of query answers.

Source selection is a time and money efficient move to make for query optimisation. It has been well studied on account of its accuracy and time-efficiency. It is out of the scope of this work, however, and so we will apply the state-of-the-art selection method in CostFed (Saleem et al., 2018) for the implementation of the Free Market. However, other than applying the optimisation methods of query planning which aim at reducing the execution time, the query planning phase in a local query engine aims at generating an optimal query plan based on the pricing function of the selected relevant sources, and the budget and preferences/priorities constraints with respect to the above execution mode of the local query engine. This is challenging for existing approaches of query planning, and we thus aim at solving the above questions with price-based query planning approaches for a local query engine to satisfy the data requirements of buyers in this work.

5.3 Summary

In this chapter, we have provided a definition of the Free Market, as an open business environment for all kinds of potential sellers and buyers. Sellers reserve the management of their data sources in terms of access control and pricing functions, while the marketplace in the Free Market only makes an announcement of the available data sources in the market. Buyers independently search for answers for their requirements and directly trade data with sellers via a local query engine. To satisfy the data requirement of a buyer, the local query engine works in a different way from the remote query engine located in marketplaces and faces the challenge of optimising query plans to meet buyers' constraints regarding budgets and preferences/priorities in their requirements.

In the following chapters, we will model the above challenge into a minimum spanning tree problem with the cost of data requests as the weight of spanning trees, introduce our heuristic approaches to approximate the optimal query plan for a local query engine and evaluate their performance in comparison to the state-of-the-art query planning algorithm. After that, we will investigate price-based query planning with budget constraints in an effort to optimise the time efficiency of query plans without exceeding a limited budget. For this, we demonstrate an approximate algorithm as a solution.

Chapter 6

Price-based Query Planning in the Free Market

With respect to the data requirements of buyers and the query execution mode of a local query engine, we aim to improve the query planning phase in a local query engine based on the price of data sources and the constraints of budgets and preferences/priorities of requirements to trade data in the Free Market. In this chapter, we first illustrate the problem with an instance of a SPARQL query over distributed RDF data, formalise the problem into a minimum spanning tree optimisation question with our tree-structured model for query plans and methods to estimate costs, and then demonstrate two heuristic algorithms with an evaluation of their performance against to the state-of-the-art query planning algorithm.

6.1 Problem Statement

Without involving pricing functions in query optimisation, the existing query planning methods only target minimising join cardinality, which leads to a waste of money with respect to the prices of data. For instance, John has a query where he wants to know Barack Obama's party membership and the party's news pages from the New York Times. However, since there is no single data source that can serve John with the answers he needs, he has to query multiple sources at the same time. Luckily, John is smart and has a background of RDF and SPARQL knowledge which were learned from W3C (W3C) as detailed in Chapter 3

Figure 6.1(a) shows John's query in the form of SPARQL with three triple patterns tp_1 , tp_2 and tp_3 . There are two data sources, DBpedia and NY Times in the Free Market that are relevant to John's query, as Figure 6.1(b) presented, and the number of triples

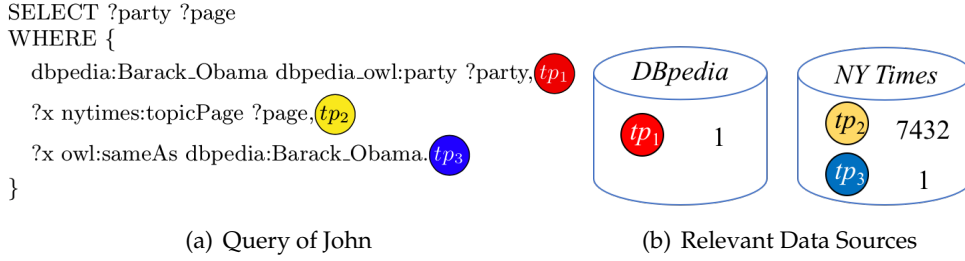


FIGURE 6.1: Query CD2 and Its Relevant Data Sources from FedBench

matching to the 3 triple patterns from their relevant data sources are 1, 7,432 and 1, respectively.

Recall Hypothesis 3.2.1 in Chapter 3. Without thinking about the cost of query answers, John would decompose his query into three sub-queries, each one including one of tp_1 , tp_2 and tp_3 , and formulate a query plan as follows:

1. Query data source, Dbpedia, by SQ1:

```
SELECT * WHERE{ dbpedia:Barack_Obama dbpedia_owl:party ?party. }
```

2. Query data source, NYTimes, by SQ2:

```
SELECT * WHERE{ ?x nytimes:topicPage ?page. }
```

3. Query data source, NYTimes, by SQ3:

```
SELECT * WHERE{ ?x owl:sameAs dbpedia:Barack_Obama. }
```

4. Join all the above intermediate results and get the answer to his original query.

Assume that the data sources, DBpedia and NYTimes, have simply decided to charge 0.01 and 0.05 per query answer, respectively. Then, John will get all the $1 + 7,432 + 1$ answers by Step 1, 2 and 3, then join them into the final answers for his original query in Step 4, which will cost him $1 \times 0.01 + 7,432 \times 0.05 + 1 \times 0.05 = 371.66$ in total.

Naturally, John wonders whether there is a cheaper way to get the same answers. If he had a plan as follows:

1. Query data source, Dbpedia, by SQ1:

```
SELECT * WHERE{ dbpedia:Barack_Obama dbpedia_owl:party ?party. }
```

2. Query data source, NYTimes, by SQ3:

```
SELECT * WHERE{ ?x owl:sameAs dbpedia:Barack_Obama. }
```

3. For each value i of $?x$ collected from the above return results, query data source, NYTimes, by updated SQ2:

```
SELECT * WHERE{ i nytimes:topicPage ?page. }
```

4. Join all the above intermediate results and get the answer for his original query.

This first asks Dbpedia with SQ1, requests answers to SQ3 from NYTimes, then continues to ask NYTimes about SQ2 after applying the values from intermediate results of SQ3 to variables in SQ2, and then join the returned results at last. This way, the cost of the answers reduces to $1 \times 0.01 + 1 \times 0.05 + 1 \times 0.05 = 0.11$ when the number of matching triples to SQ2 after joining with the known values to its variables reduces sharply from 7,432 to 1. Moreover, when the data sources change the methods of charging query answers, one possible result is that the first way of asking the sub-questions may cost the same as the second method. For example, when DBpedia and NYTimes all charge a fixed price per query, thereby, the order of the above querying and joining process will not affect the cost of the answers.

In this work, we denote the number of matching triples to a sub-query from a data source as the selectivity of the sub-query over the data source. We call the selectivity of a sub-query after joining with another sub-query as the join-selectivity of the sub-query over a data source. Therefore, the optimum cost-saving query plan for buyers is dependent on: (1) the pricing functions of data sources, (2) the selectivity of the buyer's query in relevant data sources, and (3) the join-selectivity among sub-queries. Without involving pricing functions in query optimisation, existing query planning methods only target minimising the join-selectivity, which leads to a waste of money as explained in our example. In this chapter, we focus on finding a less expensive way to collect query answers based on the pricing functions of data sources and demonstrate the impact of the above three factors in query planning in the Free Market.

6.2 Price-Based Query Planning

To formalise the problem of price-based query planning, we introduce a method to model SPARQL queries as complete graphs and use their spanning trees to represent the query plans of the original queries, and then cast the problem into a minimum cost spanning tree problem by estimating the cost of sub-queries as the weight of adding the triple pattern in the graph. We provide two approaches to estimate the cost of query plans, and then implement them within greedy algorithms to approximate the optimal cost-efficient query plan.

6.2.1 Query Plan-Tree

As a foundation to price-based query planning, we first define and present the basic concepts of Complete Query Graph and Spanning Plan-Tree. To simplify the notation of SPARQL queries, let $Q = (pVar, BGP)$ denote a SPARQL query that consists of a set of k variables, $pVar = \{v_1, v_2, \dots, v_k\}$, and a basic graph pattern (W3C) formed by triple patterns in the query, $BGP = \{tp_1, tp_2, \dots, tp_l\}$. Let $G \models Q$ denote the complete query

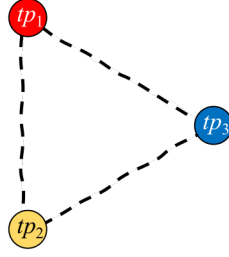


FIGURE 6.2: Complete Query Graph of John's Query

graph corresponding to Q where every single node n in G represents a triple pattern $tp \in BGP$, and each link between two nodes in G indicates that the triples matching the two triple patterns can join together, as described in Definition 6.1, where we directly use tp as n . Figure 6.2 presents the complete query graph of John's query in Figure 6.1(a), where three nodes are marked with their corresponding triple patterns and the links among them indicate the possibility of joining the results of the sub-queries that constructed by the triple patterns. Note that the possibility of joining represented by a link between two nodes stands for the possibility that the local query engine plans to execute the query by joining the corresponding sub-queries.

Definition 6.1. Complete Query Graph: A graph $G = (N, E)$ is the *query graph* of query $Q = (pVar, BGP)$, notated as $G \models Q$, where $N = \{n | n = tp, tp \in BGP\}$ and $E = \{e | e = (n_i, n_j), n_i \in N, n_j \in N, i \neq j\}$.

A query plan of a SPARQL query describes the processing order and executing destinations of sub-queries. A spanning tree of a graph is a subgraph that has all the nodes linked by a minimum number of edges and no circles. Regarding the fact that the data sources that sub-queries go to are decided by the source selection and the one-to-one correspondence between sub-queries and triple patterns of a SPARQL query, the spanning progress that a spanning tree \mathcal{T} of a complete query graph G of a SPARQL query Q is able to illustrate the process of a query plan of Q . Therefore, we refer to a spanning tree of a complete query graph as a *Spanning Plan-Tree* in Definition 6.2, where D denotes the full set of available data sources in the Free Market.

To map the process through which a tree spans a node in a complete query graph to the process that incrementally joins a triple pattern to a query plan, we use the following notations: when a tree spans a node, let it first choose a set of data sources that the corresponding sub-query goes to, which should be a subset of the selected sources of tp if the optimisation decides not to access all the selected sources; then, let $\mathcal{T}[n_i] \rightarrow n_j$ denote the spanning process that a tree $\mathcal{T} = (\tilde{N}, \tilde{E})$ spans node n_j via the edge $e < n_i, n_j >$ where $n_i \in \tilde{N}, n_j \notin \tilde{N}$.

Definition 6.2. Spanning Plan-Tree: A spanning plan-tree \mathcal{T} is a spanning tree of G where $G \models Q$ and \mathcal{T} represents a query plan of Q .

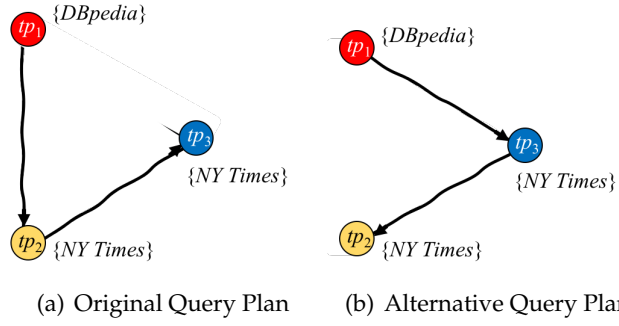


FIGURE 6.3: Spanning Plan-Trees Modelling the Two Query Plans of John's Query

The left part of Figure 6.3 lists the spanning plan-trees of the complete query graph of John's query in Figure 6.2. The arrows on edges indicate the spanning order, i.e. the join order of the three sub-queries constructed by the triple patterns, and the annotated information besides triple pattern nodes are the set of chosen data sources from Figure 6.1(b) to ask for intermediate results matching the triple patterns. The two spanning plan-trees correspond to the original and alternative query plans generated by John in the previous section. Comparing edges $e < tp_2, tp_3 >$ and $e < tp_3, tp_2 >$, which correspond to the third step in the two query plans, they are the ones that lead to the differences of thousands of times in the costs of query answers.

6.2.2 Cost Estimation of Plan-Tree

Assume each data source d has a pricing function for charging a request to access a set of triples. Note that, just as the joining order changes the number of intermediate results, the cost of a query plan changes according to the order that it incrementally adds triple patterns from a complete query graph, which means the cost of incrementally joining a triple pattern is not constant. In order to calculate the cost of incrementally adding a triple pattern, which means the cost of collecting the matching triples to the triple pattern over the chosen data sources, we estimate the selectivity of each spanning process which indicates the join-selectivity that the triple pattern considered to span based on the previous intermediate results.

Since the intermediate results returned to a sub-query lists the items matching the variables in the sub-query instead of the triples that derive the results, we decide to calculate the number of matching items to each variable in an original query if a query engine executes the query according to a query plan. Thus, we can estimate the number of triples that are used or accessed for intermediate results and the price of the query plan. To track the join-selectivity variation of triple patterns in a query $Q = (pVar, BGP)$, let $\delta(tp)$ denote the set of variables in a triple pattern tp , $amount(v, D)$ represent the number of different terms over a set of data source, D , that are matched by variable $v \in pVar$. Let $\mathcal{L}(tp)$ be the chosen data sources to ask for tuples matching tp and

$$\begin{aligned}
& \text{Join-Calculation}(\Pi(\mathcal{T}), \text{EstimatedAmount}(tp_j)) = \\
& \left(\begin{array}{l} \text{amount}(v_1, D' \cup \mathcal{L}(tp_j)) = \begin{cases} \min \{ \text{amount}(v_1, D'), \text{amount}(v_1, \mathcal{L}(tp_j)) \} & \text{amount}(v_1, D') \times \text{amount}(v_1, \mathcal{L}(tp_j)) \neq 0 \\ \max \{ \text{amount}(v_1, D'), \text{amount}(v_1, \mathcal{L}(tp_j)) \} & \text{amount}(v_1, D') \times \text{amount}(v_1, \mathcal{L}(tp_j)) = 0 \end{cases} \\ \\ \text{amount}(v_2, D' \cup \mathcal{L}(tp_j)) = \begin{cases} \min \{ \text{amount}(v_2, D'), \text{amount}(v_2, \mathcal{L}(tp_j)) \} & \text{amount}(v_2, D') \times \text{amount}(v_2, \mathcal{L}(tp_j)) \neq 0 \\ \max \{ \text{amount}(v_2, D'), \text{amount}(v_2, \mathcal{L}(tp_j)) \} & \text{amount}(v_2, D') \times \text{amount}(v_2, \mathcal{L}(tp_j)) = 0 \end{cases} \\ \\ \dots, \\ \text{amount}(v_k, D' \cup \mathcal{L}(tp_j)) = \begin{cases} \min \{ \text{amount}(v_k, D'), \text{amount}(v_k, \mathcal{L}(tp_j)) \} & \text{amount}(v_k, D') \times \text{amount}(v_k, \mathcal{L}(tp_j)) \neq 0 \\ \max \{ \text{amount}(v_k, D'), \text{amount}(v_k, \mathcal{L}(tp_j)) \} & \text{amount}(v_k, D') \times \text{amount}(v_k, \mathcal{L}(tp_j)) = 0 \end{cases} \end{array} \right)
\end{aligned}$$

FIGURE 6.4: Function for Cardinality Estimation of Joining An Intermediate Tree and A Triple Pattern Node

$\mathcal{O}(tp, d)$ denote the set of triples that tp matched in data source d . Then, the vector

$$\Pi(\mathcal{T}) = \langle \text{amount}(v_1, D'), \text{amount}(v_2, D'), \dots, \text{amount}(v_k, D') \rangle \quad (6.1)$$

where $v_i \in pVar, D' = \bigcup_{tp \in \tilde{N}} \mathcal{L}(tp)$, represents the intermediate selectivity of all the variables in the query according to a tree $\mathcal{T} = (\tilde{N}, \tilde{E})$. The estimated selectivity of tp is a vector, $\text{EstimatedAmount}(tp) = \langle \text{amount}(v_1, \mathcal{L}(tp)), \dots, \text{amount}(v_x, \mathcal{L}(tp)) \rangle$, where $v \in \delta(tp)$.

To estimate the number of items matching each variable after joining the intermediate results of \mathcal{T} and a triple pattern, we separate the variables into two categories: join variables and non-join variables. The join variables have matched items according to the current \mathcal{T} and the triple pattern, and the join process will filter and only keep the same listed items. This implies that the number of the items matched to the join variables will be no more than the number of listed items according to either \mathcal{T} or tp . The non-join variables only have matched items from the estimated results of \mathcal{T} or tp , therefore, the join process will not affect these numbers. Thereby, the intermediate join-selectivity, $\Pi(\mathcal{T} \bowtie tp_j)$, that the tree \mathcal{T} connects with a new node via edge $e < n_i, n_j >$ where $n_i \in \tilde{N}$ but $n_j \notin \tilde{N}$, is

$$\Pi(\mathcal{T} \bowtie tp_j) = \text{Join-Calculation}(\Pi(\mathcal{T}), \text{EstimatedAmount}(tp_j)) \quad (6.2)$$

where $\text{Join-Calculation}(\Pi(\mathcal{T}), \text{EstimatedAmount}(tp_j))$ is the function defined by the equation in Figure 6.4. The function calculates the join-selectivity of all variables individually. For sharing variables in \mathcal{T} and tp_j which is found by $\text{amount}(v, D') \times \text{amount}(v, \mathcal{L}(tp_j)) \neq 0$, the join-selectivity of them is not greater than the minimum selectivity of both \mathcal{T} and tp_j before the join process. As for other variables that either $\text{amount}(v, \mathcal{L}(tp_j)) = 0$ or $\text{amount}(v, D') = 0$, the selectivity will remain the same.

For the example spanning plan-tree in Figure 6.3, we list the corresponding progress in Figure 6.5, illustrating the individual spanning process from an empty tree to a query plan tree and the join process. According to the original query plan-tree in Figure 6.3(a), the first triple pattern to span is tp_1 with its $\text{EstimatedAmount}(tp) = (1, 0, 0)$, then

$$\begin{aligned} \mathcal{T}_0 &= \emptyset \\ \mathcal{T}_1 &= (N_1, E_1), N_1 = \{tp_1\}, E_1 = \emptyset, \\ \text{EstimatedAmount}(tp_1) &= \langle \text{amount}(\text{?party}, \{\text{DBpedia}\}) = 1, \text{amount}(\text{?x}, \emptyset) = 0, \text{amount}(\text{?page}, \emptyset) = 0 \rangle, \\ \Pi(\mathcal{T}_0 \bowtie tp_1) &= \text{EstimatedAmount}(tp_1) \\ &= \langle \text{amount}(\text{?party}, \{\text{DBpedia}\}) = 1, \text{amount}(\text{?x}, \emptyset) = 0, \text{amount}(\text{?page}, \emptyset) = 0 \rangle \end{aligned}$$

$$\begin{aligned} \mathcal{T}_2 &= (N_2, E_2), N_2 = \{tp_1, tp_2\}, E_2 = \{e < tp_1, tp_2 >\}, \\ \text{EstimatedAmount}(tp_2) &= \langle \text{amount}(\text{?party}, \emptyset) = 0, \text{amount}(\text{?x}, \{\text{NYTimes}\}) = 7432, \text{amount}(\text{?page}, \emptyset) = 7432 \rangle, \\ \Pi(\mathcal{T}_1 \bowtie tp_2) &= \text{Join} - \text{Calculation}(\mathcal{T}_1, \text{EstimatedAmount}(tp_2)) \\ &= \langle \text{amount}(\text{?party}, \{\text{DBpedia}\}) = 1, \text{amount}(\text{?x}, \{\text{NYTimes}\}) = 7432, \text{amount}(\text{?page}, \emptyset) = 7432 \rangle \end{aligned}$$

$$\begin{aligned} \mathcal{T}_3 &= (N_3, E_3), N_3 = \{tp_1, tp_2, tp_3\}, E_3 = \{e < tp_1, tp_2 >, e < tp_2, tp_3 >\}, \\ \text{EstimatedAmount}(tp_3) &= \langle \text{amount}(\text{?party}, \emptyset) = 0, \text{amount}(\text{?x}, \{\text{NYTimes}\}) = 1, \text{amount}(\text{?page}, \{\text{NYTimes}\}) = 0 \rangle \\ \Pi(\mathcal{T}_2 \bowtie tp_3) &= \text{Join} - \text{Calculation}(\mathcal{T}_2, \text{EstimatedAmount}(tp_3)) \\ &= \langle \text{amount}(\text{?party}, \{\text{DBpedia}\}) = 1, \text{amount}(\text{?x}, \{\text{NYTimes}\}) = 1, \text{amount}(\text{?page}, \{\text{NYTimes}\}) = 1 \rangle \end{aligned}$$

(a) Progress of Original Query Plan-Tree

$$\begin{aligned} \mathcal{T}_0 &= \emptyset \\ \mathcal{T}_1 &= (N_1, E_1), N_1 = \{tp_1\}, E_1 = \emptyset, \\ \text{EstimatedAmount}(tp_1) &= \langle \text{amount}(\text{?party}, \{\text{DBpedia}\}) = 1, \text{amount}(\text{?x}, \emptyset) = 0, \text{amount}(\text{?page}, \emptyset) = 0 \rangle, \\ \Pi(\mathcal{T}_0 \bowtie tp_1) &= \text{EstimatedAmount}(tp_1) \\ &= \langle \text{amount}(\text{?party}, \{\text{DBpedia}\}) = 1, \text{amount}(\text{?x}, \emptyset) = 0, \text{amount}(\text{?page}, \emptyset) = 0 \rangle \end{aligned}$$

$$\begin{aligned} \mathcal{T}_2 &= (N_2, E_2), N_2 = \{tp_1, tp_3\}, E_2 = \{e < tp_1, tp_3 >\}, \\ \text{EstimatedAmount}(tp_3) &= \langle \text{amount}(\text{?party}, \emptyset) = 0, \text{amount}(\text{?x}, \{\text{NYTimes}\}) = 1, \text{amount}(\text{?page}, \emptyset) = 0 \rangle, \\ \Pi(\mathcal{T}_1 \bowtie tp_3) &= \text{Join} - \text{Calculation}(\mathcal{T}_1, \text{EstimatedAmount}(tp_3)) \\ &= \langle \text{amount}(\text{?party}, \{\text{DBpedia}\}) = 1, \text{amount}(\text{?x}, \{\text{NYTimes}\}) = 1, \text{amount}(\text{?page}, \emptyset) = 0 \rangle \end{aligned}$$

$$\begin{aligned} \mathcal{T}_3 &= (N_3, E_3), N_3 = \{tp_1, tp_3, tp_2\}, E_3 = \{e < tp_1, tp_3 >, e < tp_3, tp_2 >\}, \\ \text{EstimatedAmount}(tp_2) &= \langle \text{amount}(\text{?party}, \emptyset) = 0, \text{amount}(\text{?x}, \{\text{NYTimes}\}) = 7432, \\ &\quad \text{amount}(\text{?page}, \{\text{NYTimes}\}) = 7432 \rangle, \\ \Pi(\mathcal{T}_2 \bowtie tp_2) &= \text{Join} - \text{Calculation}(\mathcal{T}_2, \text{EstimatedAmount}(tp_2)) \\ &= \langle \text{amount}(\text{?party}, \{\text{DBpedia}\}) = 1, \text{amount}(\text{?x}, \{\text{NYTimes}\}) = 1, \text{amount}(\text{?page}, \{\text{NYTimes}\}) = 1 \rangle \end{aligned}$$

(b) Progress of Alternative Query Plan-Tree

FIGURE 6.5: Progress of Incrementally Adding Triple Patterns into Query Plan-Trees in Figure 6.3

span triple pattern tp_2 via edge $e < tp_1, tp_2 >$ following which is the join process $\text{Join} - \text{Calculation}(\mathcal{T}_1 \bowtie tp_2)$; last, span triple pattern tp_3 by joining \mathcal{T}_2 and tp_3 with a result of $\Pi(\mathcal{T}_2 \bowtie tp_3) = (1, 1, 1)$.

Due to the possibility of multiple data sources chose for one sub-query and the price variation of different data sources, to estimate the cost of adding a triple pattern to an intermediate tree, we further investigate the selectivity and join-selectivity of triple patterns from individual data sources and then apply its pricing function individually to calculate the estimated cost. Regarding the previously known information about data sources, we provide the following for the cost estimation approach: general cost estimation and summary-based cost estimation, which are designed, respectively, for where there is zero knowledge about data sources and where there is knowledge about summaries of data sources.

6.2.2.1 General Cost Estimation

We measure the join-selectivity of a triple pattern, tp by the selectivity of variables in tp after joining it with a tree $\mathcal{T} = (\tilde{N}, \tilde{E})$ with tp :

$$I = \prod_{v \in \delta(tp)} \text{amount}(v, D''), \text{ amount}(v, D'') \in \Pi(\mathcal{T} \bowtie tp). \quad (6.3)$$

where $D'' = D' \cup \mathcal{L}(tp_j)$. It multiplies the number of items matching each variable to estimate the number of triples that derive the intermediate results. This is because, without further information about the data sources, we cannot adopt any possible approach to approximate the real number of the triples but use this method instead to calculate the maximum possible number. Hence, the general estimated number of the join-selectivity of tp from a data source d is

$$I(d) = \prod_{v \in \delta(tp)} \text{amount}(v, D' \cup \{d\}). \quad (6.4)$$

where $\text{amount}(v, D' \cup \{d\}) \in \text{Join} - \text{Calculation}(\mathcal{T} \bowtie \text{EstimatedAmount}[d](tp))$, and $\text{EstimatedAmount}[d](tp) = \langle \text{amount}(v_1, \{d\}), \dots, \text{amount}(v_x, \{d\}) \rangle, v_i \in \delta(tp)$ denote the selectivity of tp from d . Then, the estimated cost of joining triple pattern node tp in order to query data source d is $PF[d](I(d))$, where $PF[d]$ notates the pricing function of d . And the cost for a tree \mathcal{T} to join tp over all chosen data sources in $\mathcal{L}(tp)$ is

$$c(tp) = \sum_{d \in \mathcal{L}(tp)} PF[d](I(d)). \quad (6.5)$$

The above estimation takes all the possible data sources as a candidate to purchase the intermediate results of sub-queries, and then the estimated cost of spanning an edge to join a triple pattern is the upper bound of the real cost.

For the spanning process in Figure 6.5 of the plan-trees in Figure 6.3, the costs of spanning node tp_2 are

- Original Plan:

$$\begin{aligned} c(tp_2) &= PF[\text{NYTimes}](I(\text{NYTimes})) \\ &= PF[\text{NYTimes}](\text{amount}(?x, \{\text{NYTimes}\}) \times \text{amount}(?page, \{\text{NYTimes}\})) \\ &= PF[\text{NYTimes}](7,432 \times 7,432) \\ &= PF[\text{NYTimes}](55234623); \end{aligned}$$

- Alternative Plan:

$$\begin{aligned}
c(tp_2) &= PF[NYTimes](I(NYTimes)) \\
&= PF[NYTimes](comp?x, \{NYTimes\}) \times comp?page, \{NYTimes\}) \\
&= PF[NYTimes](1 \times 1) \\
&= PF[NYTimes](1).
\end{aligned}$$

6.2.2.2 Summary-based Cost Estimation

When summaries of the data sources are available in the announcement of a market-place in the Free Market, as described in Chapter 5, we can use those summaries to improve the above-mentioned estimation. Instead of estimating the join-selectivity of all variables, with summaries, we can calculate the number of triples matching a triple pattern from one data source so as to estimate the cost of the triple pattern. Following the notations in [Saleem et al. \(2018\)](#), the calculated join-selectivity of joining a triple pattern, tp , to a tree, $\mathcal{T} = (\tilde{N}, \tilde{E})$, by querying data source d based on the summary statistics of d is:

$$I^*(d) = \begin{cases} M(tp, d) \times \min \left\{ amount'(s, D'), amount(s, \{d\}) \right\} & \text{if } !b(s) \wedge j(s), amount'(s, D') \in \Pi(\mathcal{T}) \\ M(tp, d) \times \min \left\{ amount'(o, D'), amount(o, \{d\}) \right\} & \text{if } !b(o) \wedge j(o), amount'(o, D') \in \Pi(\mathcal{T}) \\ M(tp, d) \times Card(tp, d) & \text{otherwise.} \end{cases} \quad (6.6)$$

where $M(tp, d)$ is the average frequency of a multi-valued predicate of data source d and $Card(tp, d)$ is the result number of tp for querying data sources d , which are updated from the definition of $M(tp)$ and $C(tp)$ in CostFed ([Saleem et al., 2018](#)). Meanwhile, $b()$ is a Boolean judgement of whether the subject s or object o in tp is a variable or not, and $j()$ is a Boolean judgement of whether the results are joined by subject s or object o in tp or not. $amount(s, \{d\})$ and $amount(o, \{d\})$ is the distinct selectivity of s and o in data source d from statistical summaries.

For the example, for the following query and data sources, after querying tp_1 at d , there are only 2 results for tp_2 in d which can be joined with the previous results of tp_1 . In our estimation, the cardinality of joining tp_2 to tp_1 on querying d is $M(tp_2, d) \times \min\{comps, D'\}, amount(s, \{d\})\} = 2/1 \times \min\{2, 1\} = 2$.

<pre> SELECT * WHERE { ?s p_1 ?o_1. (tp1) ?s p_2 ?o_2. (tp2) } </pre>	<p>d:</p> <pre> subject_1 p_1 object_1, object_2, object_3 subject_1 p_2 object_4, object_5 subject_2 p_1 object_6 </pre>
---	--

Therefore, the estimated cost based on summary statistics of a triple pattern is:

$$c(tp) = \sum_{d \in \mathcal{L}(tp)} PF[d](I^*(d)). \quad (6.7)$$

6.2.3 Minimum Cost Spanning Plan-Tree

When all edges in a graph have assigned weights, a minimum spanning tree (MST) marks the spanning trees with the minimum possible total edge weight. Hence, by casting the cost of joining a triple pattern as the weight of the corresponding node to span in a complete query graph of a query, the price-based query planning optimisation problem is problem \mathcal{X} : Find a minimum spanning plan-tree of the complete query graph with an estimated cost of spanning a triple pattern as the weight of the corresponding edge where the minimum spanning plan-tree represents a query plan to collect query answers with the lowest cost.

Existing well-known MST algorithms, *Boruvka* (Boruvka, 1926), *Prim* (Prim, 1957) or *Kruskal* (Kruskal, 1956), are all greedy algorithms that have been proved in finding an optimum solution in cases of constant weights or costs of edges. However, based on the previous discussion of estimated costs, it is evident that the costs of spanning an edge are not constant but rather change according to the spanning process. It invalidates the optimisation of the existing MST algorithms in finding a minimum cost spanning plan-tree but approximating the optimal solutions.

We attempt to approximate the optimal query plans which have the minimum cost by a greedy algorithm with the adoption of the above-mentioned two cost estimation methods in Equation 6.5 and 6.7 for the calculation of $c(tp)$, and we name them as a general greedy (Gen-Greedy) algorithm and a summary-based greedy (Sum-Greedy) algorithm. As detailed in Algorithm 3, we start the algorithms by modelling the input query into a complete query graph G (Line 1), initialising an empty tree whose cost is zero and the join-selectivity vector is empty (Lines 2-3), and we name a temporary cost variable with an infinite value (Line 4). To choose the first node, we iterate each node in G and span the empty tree with the node that has the lowest cost (Lines 5-13). After that, the loop of the spanning progress of the tree begins until the tree has spanned all nodes in G (Line 14-43). Each loop of the spanning process iterates individual edges in G (Line 18), such that we first get the corresponding triple patterns to the nodes of an edge (Line 21-22), analyse their existence in the tree (Line 21 and Line 29) and calculate their costs (Line 22 and Line 30), and then find the triple pattern whose corresponding node does not exist in the tree and costs less than all the others (Lines 23-27 and Lines 31-35) to span (Lines 38-41) and remove the edge from G (Line 42).

Algorithm 3: Greedy(BGP, D)**Input:** $Q = (pVar, BGP)$, a SPARQL query; D , the set of data sources.**Output:** \mathcal{T} , a tree indicating a query plan.

```

1  $G(N, E) \leftarrow G \models Q;$ 
2  $\mathcal{T} \leftarrow G(\emptyset, \emptyset);$ 
3  $C \leftarrow 0; \Pi \leftarrow \emptyset;$ 
4  $c \leftarrow \infty;$ 
5 foreach  $n$  in  $G.N$  do
6    $tp \leftarrow n;$ 
7    $c' \leftarrow c(tp);$ 
8   if  $c' < c$  then
9      $\mathcal{T}.\tilde{N} \leftarrow \emptyset; \mathcal{T}.\tilde{N}.add(n);$ 
10     $c \leftarrow c';$ 
11     $\Pi \leftarrow EstimatedAmount(tp);$ 
12  end
13 end
14 while  $\mathcal{T}.\tilde{N} \neq G.N$  do
15    $e'(n_x, n_y) \leftarrow \text{NULL};$ 
16    $c \leftarrow \infty;$ 
17    $\Pi' \leftarrow \emptyset;$ 
18   foreach  $e(n_x, n_y)$  in  $G.E$  do
19      $tp_x \leftarrow n_x;$ 
20      $tp_y \leftarrow n_y;$ 
21     if  $n_x \in \mathcal{T}.\tilde{N} \ \&\& \ n_y \notin \mathcal{T}.\tilde{N}$  then
22        $c'' \leftarrow c(tp_y);$ 
23       if  $c'' < c$  then
24          $e' \leftarrow e < tp_x, tp_y >;$ 
25          $c \leftarrow c'';$ 
26          $\Pi' \leftarrow J(\Pi, EstimatedAmount(tp_y));$ 
27       end
28     end
29     else if  $n_x \notin \mathcal{T}.\tilde{N} \ \&\& \ n_y \in \mathcal{T}.\tilde{N}$  then
30        $c'' \leftarrow c(tp_x);$ 
31       if  $c'' < c$  then
32          $e' \leftarrow e < tp_y, tp_x >;$ 
33          $c \leftarrow c'';$ 
34          $\Pi' \leftarrow J(\Pi, EstimatedAmount(tp_x));$ 
35       end
36     end
37   end
38    $\mathcal{T}.N \leftarrow \mathcal{T}.N \cup \{n \leftarrow e'.tp_y\};$ 
39    $\mathcal{T}.E \leftarrow \mathcal{T}.E \cup \{e'\};$ 
40    $C \leftarrow C + c;$ 
41    $\Pi \leftarrow \Pi';$ 
42    $G.E \leftarrow G.E - \{e'\};$ 
43 end
44 return  $\mathcal{T};$ 

```

6.3 Evaluation

The purpose of this chapter is to address the question: Can buyers purchase query answers at a lower price in the Free Market? Therefore, to evaluate the performance of the proposed algorithms, we first design a simulated Free Market. For data sources and queries in the Free Market, we use the data sources and queries from FedBench (Schmidt et al., 2011) assuming they are from independent sellers and buyers, respectively. For each data source, we prepare its summary statistics, an access interface and a pricing function, as the Free Market required in Chapter 5. In addition, the preparation of pricing functions consists of different types and parameter settings so as to enable further evaluation of the algorithms' sensitivity to data prices. For the local query engine, we adopt the query engine in CostFed Saleem et al. (2018) in our implementation. We use the parsing and source selection module in CostFed, and then use our price-based query planning algorithms to generate query plans that are then executed based on the query execution model described in Section 5.2, and record the real number of accessed triples and prices. We use this local query engine to conduct our experiments so as to demonstrate how efficiently our algorithms minimise the cost of query answers in the Free Market and evaluate the impact of pricing functions on their performance compared to the state-of-the-art query planning approach in CostFed (Saleem et al., 2018).

This section first illustrates the settings of the simulated Free Market, explains the experiment metrics to evaluate and compare the performance of query planning algorithms, and then it presents and discusses the experiment results.

6.3.1 Experiment Setup

6.3.1.1 Data Sources

- **Summary Statistics:** in this work, we apply the state-of-the-art template of data summary statistics proposed by CostFed. The summary statistics take the skew distribution of subjects and objects per predicate in each RDF data source and contain the prefixes of URIs that have been constructed to support the source selection approaches.
- **Access Interface:** We adopt Fuseki (fus) to set up a SPARQL endpoint for each data source as its query interface.
- **Pricing Function:** We choose the three widespread pricing functions from observations and literature mentioned in Chapter 2: price-per-tuple, flat rate and freemium pricing function.

Note that each intermediate result to a sub-query corresponds to a data tuple, i.e. a triple in RDF data sources because of Hypothesise 3.2.1 in Chapter 3. Hence, the above

three pricing functions assign the same price to every result, each request and open a certain number of results for free but assign the same price to every extra result, respectively.

To represent the variation of the price settings of data sources, we formalise a general price setting with 3 parameters that serve to model the 3 types of pricing functions above, and then assign random values to the parameters. Let function $f(X, L, p)$ represent the pricing settings of the data sources, where x is the fixed price to receive query results within the number of L , and p is the price of a query result after exceeding L . Then, the above three types of price function settings are as follows:

- Flat Pricing Function $f(X, \infty, 0)$: fixed price, X , for any query results;
- Freemium Pricing Function $f(X, L, p)$: fixed price, X , for a certain amount of query results, L , and charge a fixed price per extra query result, p ;
- Per Pricing Function $f(0, 0, p)$: charge per query result at the price of p ;

We arrange three classes of experiments: *Flat*, *Freemium* and *Per*, and in each class we set all the data sources with the same type of above pricing functions but randomly assign values to the parameters in each pricing function of data sources. The ranges of the variation of parameters X , L and p are $[0, 100]$, $[0, 10000]$ and $[0, 1]$, respectively. We generate 10 groups of random parameter settings for the pricing functions of the data sources in each class of experiment as listed in Table B.1, B.3 and B.2 in Appendix B.

The *Flat* experiment class is the baseline for the evaluation since the price of query answers charged by data sources is irrelevant to the amount of collected data, which indicates the performance of the 3 algorithms, Gen-Greedy, Sum-Greedy and CostFed, will be examined without the impact of pricing functions, i.e. only on the performance of runtime efficiency. With the freemium pricing functions and price-per-triple, we try to model real-world transaction cases, which is ‘try then pay’ and ‘pay as you go’. In addition, the two types of pricing functions are aiming to work on data requests at different amount levels. In other words, freemium pricing functions are expected to be friendly with data requests that ask for a large amount of data, whereas price-per-triple pricing functions reflect the variation of a single triple in the amount of the requested results and suit the collections of smaller amount of data. Thus, the experiment classes of *Freemium* and *Per* can demonstrate the impact of data price on both large data requests and small data requests.

6.3.1.2 Query

From the 25 queries of FedBench, we discard 7 queries for our experiments. The reason for this is that each of those 7 queries has only one relevant source for all the triple patterns in the query, which indicates that there is no need for the optimisation of query

TABLE 6.1: Queries of FedBench And the Entire Number of the Results. #ER: the number of entire results with Data Sources ChEBI, DBpedia, DrugBank, KEGG, Geonames, Jamendo, LinkedMDB, NY Times, SW Dog Food Indexed as #1,#2,#3,#4,#5,#6,#7,#8,#9.

	Query	#ER	#1	#2	#3	#4	#5	#6	#7	#8	#9
→	CD1	90		✓	✓		✓		✓	✓	
→	CD2	1		✓						✓	
→	CD3	2		✓						✓	
→	CD4	1							✓	✓	
→	CD5	2		✓					✓		
→	CD6	11					✓	✓		✓	✓
→	CD7	1					✓			✓	
	LD1	309									✓
	LD2	185									✓
	LD3	162									✓
	LD4	50									✓
	LD5	28		✓							
→	LD6	39		✓			✓	✓	✓		✓
→	LD7	1216					✓			✓	
→	LD8	22		✓	✓						
	LD9	1		✓							
→	LD10	3		✓						✓	
→	LD11	376		✓							✓
	LS1	1159			✓						
→	LS2	333		✓	✓						✓
→	LS3	9054		✓	✓						
→	LS4	3			✓	✓					
→	LS5	393	✓		✓	✓		✓			✓
→	LS6	28			✓	✓					
→	LS7	144			✓	✓					

plans. Table 6.1 lists all the queries of FedBench, points out those we use in our experiment by ‘→’ and marks their selected sources with ‘✓’.

6.3.1.3 Evaluation Metrics

We use the following two metrics to illustrate the performance of our price-based query planning algorithm:

- **Cost Reduction:** the extent of the discount that a client can achieve through the price-based query planning, i.e. Gen-Greedy and Sum-Greedy algorithms, compared to an existing state-of-the-art query planning algorithm that did not consider the prices of data, i.e. CostFed. The higher the cost reduction is, the more money buyers can save.
- **Time:** the runtime of optimising and executing query plans that the query engine takes to return a response to buyers after receiving a query.

We conduct our experiments on a Red Hat Enterprise Linux 7 (RHEL7) virtual machine with 4 CPU Cores and 32GiB RAM and 200GiB Storage and set up a timeout of 24 hours for every single experiment.

6.3.2 Experiment Results

In the following discussion of the experiment results, we display a portion of the complete results sufficient to compare the performance of our algorithms and illustrate the reasons for the differences in the performance. For the complete experiment results, please refer to Appendix B.3.

6.3.2.1 Cost-Efficiency

We apply Gen-Greedy, Sum-Greedy and CostFed query planning algorithms for the 18 queries with 30 different groups of price settings in 3 experiment classes, and execute the query plans generated by these to calculate the cost of the query answers. Figure 6.6 shows the reduced cost of executing the query plans generated by the Gen-Greedy and Sum-Greedy algorithm compared to CostFed. The letter in each square marks which algorithm has the lowest cost plans, whereas the colour represents the ratio of reduced costs by the algorithms to CostFed for the corresponding query (row) at the random pricing function setting (column). The mark '-' denotes that the query plans generated by the three algorithms cost the same, while 'X' indicates that the execution of the corresponding query plans is out of memory or time and thus fails to get results. For the details of reduced costs, the results of all the conducted experiments are given in Appendix B.4.

The comparison in Figure 6.6 displays the performance of the Gen-Greedy and Sum-Greedy algorithms in generating cost-efficient query plans. Either Gen-Greedy or Sum-Greedy reduces the cost of query answers compared to CostFed in most experiment settings for 6 queries, and performs the same with CostFed for 12 queries, although the cost of query answers to Query CD6 is higher compared to CostFed. Specifically, half of the experiment results prove that the three algorithms have the same performance under various pricing function settings, especially for flat pricing functions. Moreover, both the Gen-Greedy and Sum-Greedy algorithms accomplish the goal of reducing the costs of query answers for the cross-domain (CD) queries CD1, CD2, CD3 and CD5 under the setting of Freemium and Per pricing functions, especially CD2, whose query answer costs are reduced to just a quarter of those of CostFed. Sum-Greedy reduces the cost of the query answers to CD5 to 50% compared to CostFed, whereas Gen-Greedy performs slightly better for LD11 under Per pricing functions.

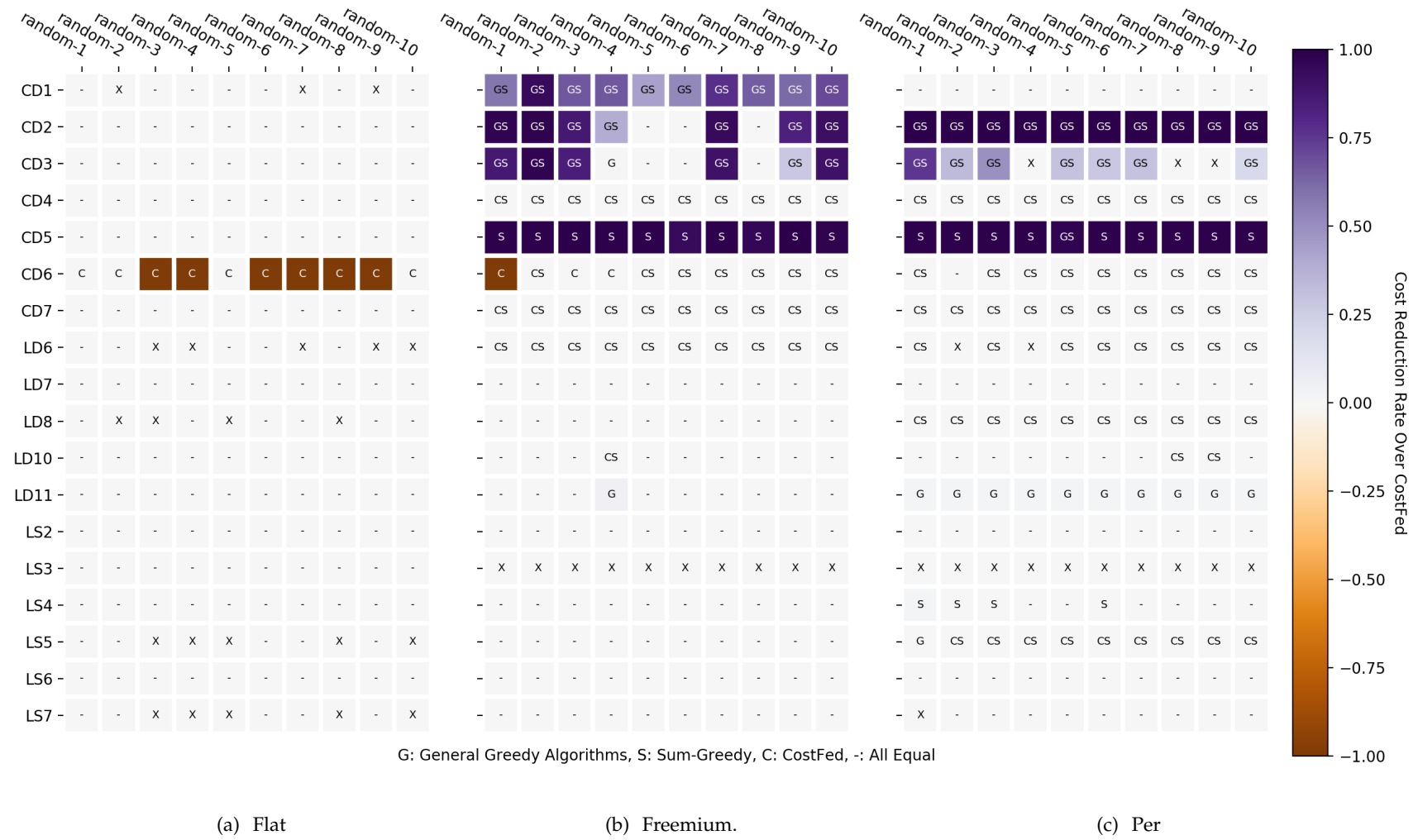


FIGURE 6.6: Cost Reduction Comparison over Random Pricing Function Settings

In the following sections, to demonstrate the reasons behind the performance, we will compare the query plans generated by Gen-Greedy and Sum-Greedy against CostFed individually to understand the impact of queries and data on the performance and further examine their performance sensitivity to prices.

6.3.2.2 Impact of Query and Data

Considering the cost estimation calculation in both the Gen-Greedy and Sum-Greedy algorithms, the accuracy of the estimation is related to the query structure and its relevant data sources. Hence, an analysis of queries and data is necessary to understand and compare the performance of the algorithms in reducing costs.

CostFed vs Gen-Greedy Theoretically, since the Gen-Greedy algorithm adopts the estimation of upper bound data costs, any circumstances that lift the upper bound estimation higher than the real costs will lead to a worse performance of Gen-Greedy in reducing costs, and vice versa. These kinds of circumstances can be the results of varied pricing functions or the structure of query and data. We will first explain the impact of query and data and leave the analysis of price impact for the next section.

One simple circumstance is that the join-selectivity of triple patterns among data sources is far more different than their selectivity over data sources. Because the estimation calculation in Gen-Greedy only uses the smallest selectivity of triple patterns to estimate join-selectivity, which means that when real join can sharply reduce the selectivity of triple patterns over data sources, the gap between Gen-Greedy's estimation and the real value can be enormous, and Gen-Greedy will end with bad performance. One example of this is Query LD8, whose triple patterns' join-selectivity varies sharply up to 20,000 times.

To explain this circumstance, we show the stacked costs of Query LD8 in Figure 6.7, where every coloured bar marks the join-selectivity of a triple pattern over a data source as well as its cost in the order of execution. Since *tp4* and *tp5* in Query LD8 both choose source *DBpedia* and adopt its pricing function for cost estimation, the selection of a lower cost triple pattern is the one with less join-selectivity. The selectivities of variables in *tp4* and *tp5* that are not join-able with *tp1*, *tp2* and *tp3* in Query LD8 are 1,243,722 and 449,152, respectively, while the join-selectivity if joined with *tp1*, *tp2* and *tp3*, as Figure 6.7 shows, are 7 and 53. Due to this upper bound estimation approach, Gen-Greedy prefers to join with the triple pattern that has lower selectivity, which is *tp5*, whereas the actual lower join-selectivity triple pattern is *tp4* which can also reduce the join-selectivity of *tp5* from 53 to 22, as the stacked costs of CostFed in Figure 6.7 show.

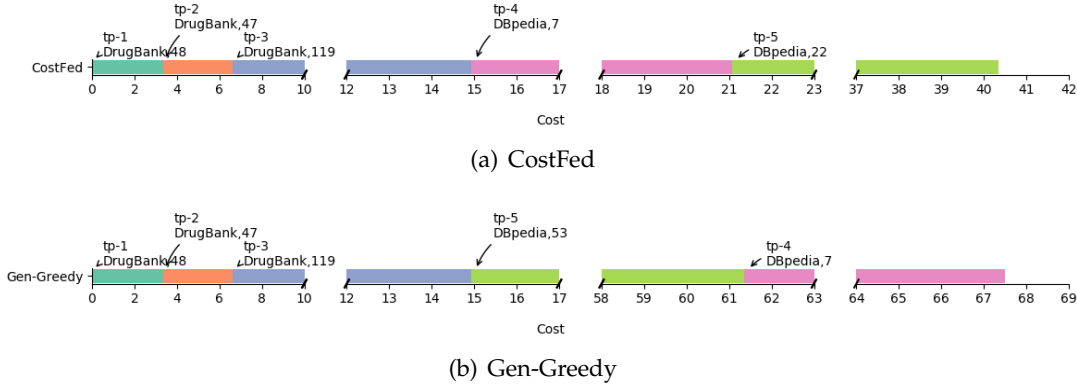


FIGURE 6.7: Stack Cost of Query Plans of Query LD8 in Group 4 of *Per* Experiment Class

Another worse circumstance is when a triple pattern in a query has multiple relevant data sources but the join-selectivity between the triple pattern and other triple patterns in the query over some of those data sources is none. In such a case, Gen-Greedy will include all the relevant data sources to estimate the upper-bound of costs without knowing the semantics of data, but, in fact, it includes the costs of collecting data from those sources who offer no join results and return empty query answers. In other words, compared to CostFed, Gen-Greedy performs badly in reducing costs when the summary statistics provides more accurate semantic join information, which is adopted in CostFed. This is particularly significant for Query CD6, as Figure 6.6 shows, and also CD7, LD6 and LD10.

To illustrate how query and data affect the performance of Gen-Greedy, Figure 6.8 shows the query plans of Query CD6 in the order of executing triple patterns at certain data sources. Each greedy spanning step ranks left triple patterns in ascending cost order and selects the first triple pattern to span. The tables beside the triple patterns display their relevant data sources and the corresponding join-selectivity over a data source. The numbers on the right side of the charts are the selectivity of the triple pattern over corresponding data sources. The selectivity of *tp1* and *tp2* of Query CD6 is not null as Figure 6.8 shows. But Gen-Greedy decides to query *tp2* first in Figure 6.8(b) rather than *tp3* as CostFed does in Figure 6.8(a). Because Gen-Greedy lacks the knowledge that the join-selectivity of *tp3* from *NY Times* is 0 and from *Geonames* sharply drops from 7479713 to 1216. This leads to the estimated cost of *tp3* is higher than *tp2*, which motivates Gen-Greedy to query *tp2* first.

To further test and verify this circumstance, we conduct experiments that remove non-join-data-sources *NYT* and *SWDFood*, and then redo the optimisation of query plans for Query CD6 by CostFed and Gen-Greedy. Figure 6.9 displays the cost of Query CD6 after executing their query plans. The y-axis of Figure 6.9 is the cost of the results and the x-axis is the number of different pricing function settings. After erasing the non-join-data-sources, all of the three algorithms are on the same level for price-based

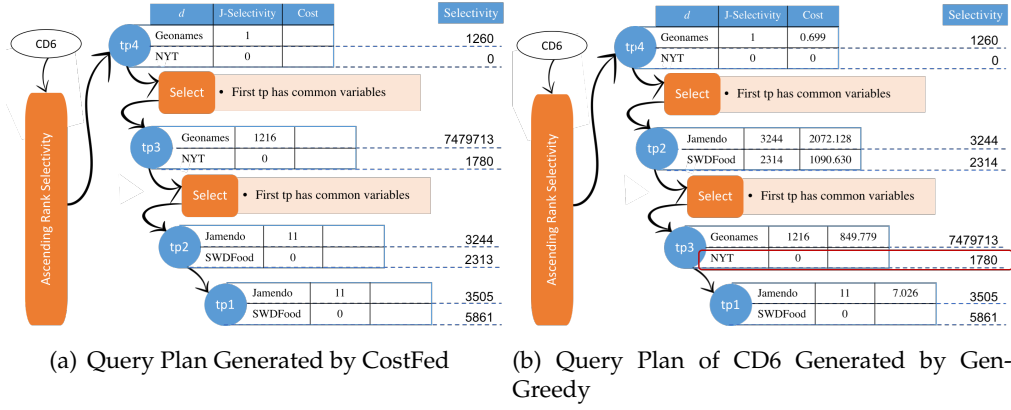


FIGURE 6.8: Query Plans of CD6 with the Group 1 Per Pricing Function Setting

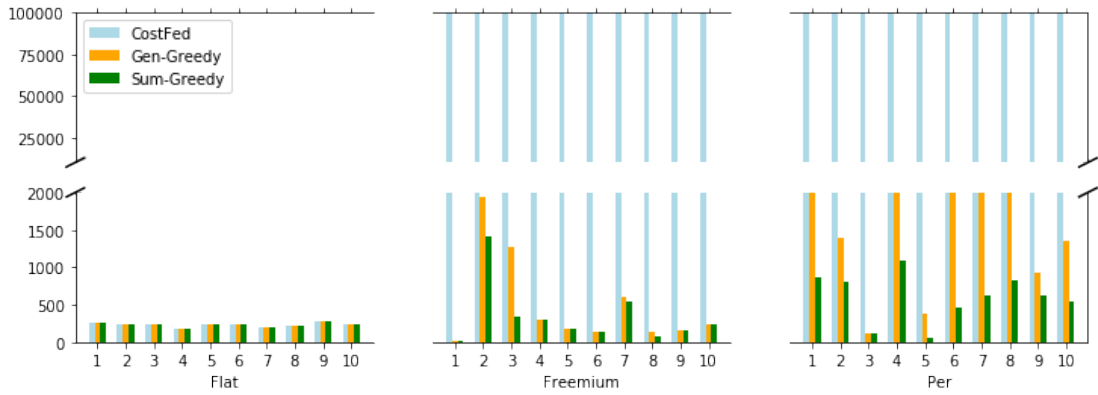


FIGURE 6.9: CD6

query planning in the *Flat* experiment class. This is expected since the data bill from the data sources is irrelevant to the amount of collected data. However, for the other 20 groups of random pricing function settings, Gen-Greedy can help buyers save more than 80% of their payments that they would pay if they used CostFed. Sum-Greedy, meanwhile, which adopts summary statistics in its estimation, performs even better than Gen-Greedy.

However, in some circumstance, summary statistics are not helpful. Since the summary-based estimation only seeks a triple pattern that requires a minimum number of triples, it may lead to processing more triples for the remaining triple patterns. Query LD11 is an instance of this circumstance and Gen-Greedy shows its advantage in this case over CostFed and Sum-Greedy. By comparing the stacked costs of query plans generated by CostFed and Gen-Greedy in Figure 6.10, we can see that CostFed and Sum-Greedy lose since their summary-based estimation does not help them discover that joining with *tp4* first would help reduce the join-selectivity of *tp3*.

Furthermore, the performance of Gen-Greedy is sensitive to the pricing functions in cases where each triple pattern in a query is only relevant to one data source. For example, at some pricing functions, Gen-Greedy reduces the cost of the results to Query

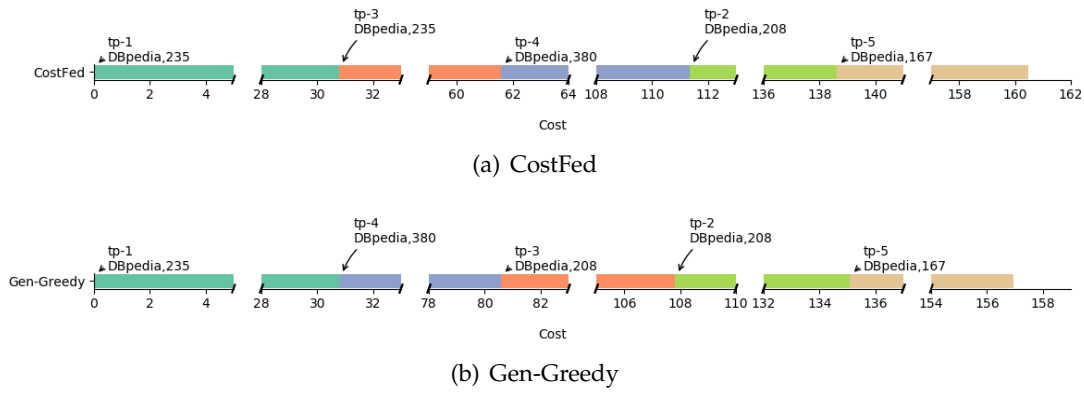


FIGURE 6.10: Stack Cost of Query Plans of Query LD11 in Group 1 *Per Experiment Class*

CD2 compared with CostFed, but also generates query plans that cost the same as CostFed. We will further compare the performance sensitivity of Gen-Greedy and the other algorithms in Section 6.3.2.3.

CostFed vs Sum-Greedy Two reasons underpin the main differences between CostFed and Sum-Greedy. The first is how they structure triple patterns. To join with triple patterns that share the same variables instead of random triple patterns to reduce intermediate join-selectivity, CostFed groups triple patterns including the same variables into a so-called *ExclusiveGroup* structure without ranking triple patterns according to their selectivity. The *ExclusiveGroup* structure helps CostFed to reduce the join computation cost but limits its ability to reorder the triple patterns among different *ExclusiveGroups*. Meanwhile, Sum-Greedy fully compares all the triple patterns individually to seek a minimum cost plan. Therefore, when a set of triple patterns in a query share a variable but their join-selectivity is not in ascending order, a query plan with minimum intermediate results, which are generated by CostFed, is not the price optimum query plan, whereas Sum-Greedy reconsiders the order of them with respect to their costs. This specific reason leads to the outperformance of Sum-Greedy versus CostFed in Query CD2, CD3 and CD5.

To fully analyse the above circumstance, we display Query CD5 in below and showcase the stacked costs of query plans of Query CD5 generated by Sum-Greedy and CostFed in Figure 6.11.

```
SELECT ?film ?director ?genre WHERE {
  ?film dbpedia-owl:director ?director .      (tp1)
  ?director dbpedia-owl:nationality dbpedia:Italy . (tp2)
  ?x owl:sameAs ?film .                      (tp3)
  ?x linkedMDB:genre ?genre .                  (tp4)
}
```

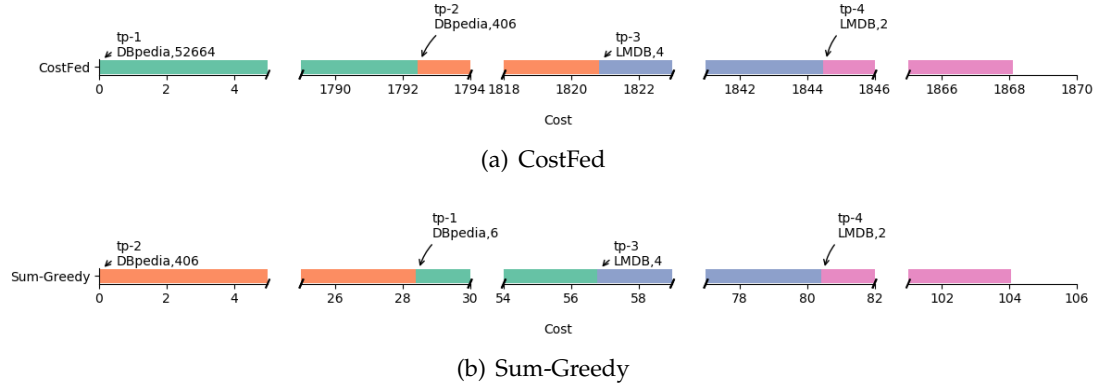


FIGURE 6.11: Stack Costs of the Query Plans of Query CD5 in Group 6 of *Freemium* Experiment Class

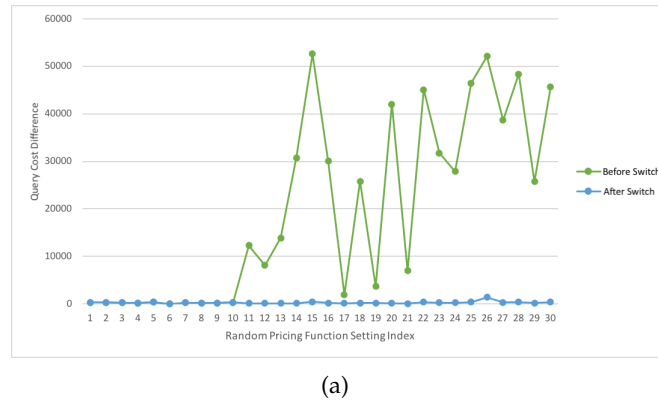


FIGURE 6.12: Experiment Results of Query CD5 After Switching Triple Patterns

It can be seen that triple pattern $tp1$ and $tp2$ in Query CD5 share the same variable, $?director$, which implies they can join together. Obviously, Sum-Greedy reconsiders the query order of $tp1$ and $tp2$ even if it does not affect the number of intermediate results of joining them together, which is the only concern CostFed has. Then, Sum-Greedy makes its decision to query the less expensive triple pattern $tp2$ first and ends up saving more than 90% of the costs charged in CostFed.

To verify this reason, we first calculate the cost differences between CostFed and Sum-Greedy (the cost of CostFed minus the cost of Sum-Greedy), and then conduct an extra experiment for Query CD5 after switching the order of triple patterns $tp1$ and $tp2$ of CD5, calculate the cost differences between Sum-Greedy and CostFed after the switch, and present the two groups of cost differences in Figure 6.12. Before the switch, Sum-Greedy costs less than CostFed for 21 different pricing function settings, but after the switch, the cost differences between the two algorithms are around zero.

By ranking triple patterns in the order of their costs, Sum-Greedy can find a cheaper query plan, such as the above case, but also can make a not-joinable data purchase, when the price of data is really cheap, and yet end with an expensive query plan. This happens when multiple cheaper data sources are relevant to triple patterns but cannot

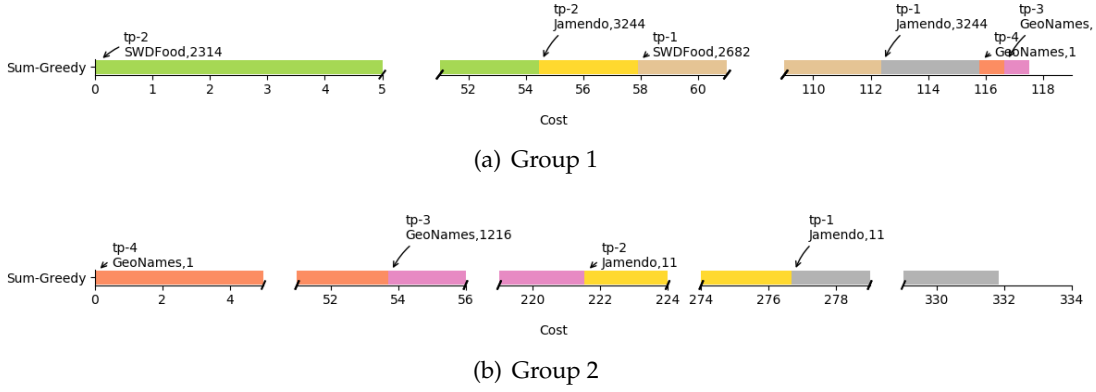


FIGURE 6.13: Stack Cost of Query Plans of Query CD6 Generated by Sum-Greedy in Group 2 of *Freemium* Experiment Class

join with others, especially when statistics do not support accurate join estimation for semantics. For example, Figure 6.13 presents the stacked costs of the query plans of Query CD6 generated by Sum-Greedy at the groups 1 and 2 in the *Freemium* experiment class. In the case of group 1, where purchasing from *Geonames* and *NY Times* is more expensive than *SWDFood* and *Jamendo* based on the estimated selectivity of *tp4* and *tp2*, Sum-Greedy chooses to query *tp2* first, leading to a purchase of 2,314 unnecessary results, which also happens to the purchase from *SWDFood* for *tp1*. However, when the pricing functions vary to group 2, where the situation is just the opposite, *tp5* is the first option for Sum-Greedy which avoids the unnecessary payments for purchasing data from *SWDFood* for both *tp1* and *tp2*.

This circumstance is similar to the one that caused the failure of Gen-Greedy for Query CD6. After removing the non-join-able data sources, Sum-Greedy takes the leading position in minimising the costs of query results, as Figure 6.9 shows. This is because both algorithms adopt the greedy algorithm which finds the local optimal MST rather than the global optimal MST when the costs of spanning an edge, i.e. a triple pattern node is not constant. In the next section, we will present a further analysis of the impact of pricing functions on both Gen-Greedy and Sum-Greedy in the next section. Note that, despite all the above reasons for the different performance of Gen-Greedy, Sum-Greedy and CostFed, the costs of the results of Query CD1, LD7, LS2 and LS6 are the best when the query plans generated by Gen-Greedy or Sum-Greedy are executed, no matter how the pricing function changes.

6.3.2.3 Price Sensitivity of Gen-Greedy and Sum-Greedy

The analysis above shows that the performance of both Gen-Greedy and Sum-Greedy in cost-efficient query planning is under the impact of the pricing functions of the data sources, and that the variation of pricing functions can enhance or weaken the efficacy of Gen-Greedy and Sum-Greedy in finding price-based query plans to reduce the costs

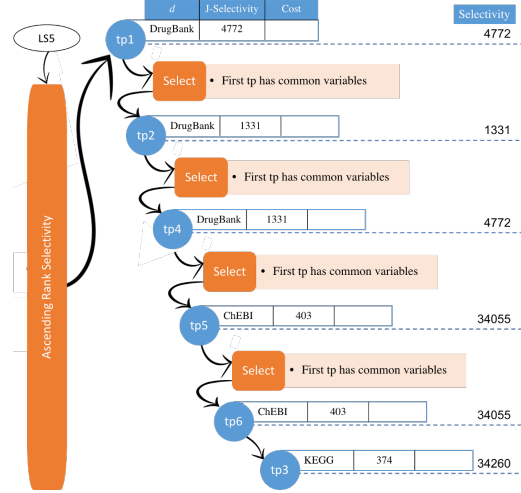


FIGURE 6.14: Query Plan of LS5 Generated by CostFed

of query results, which is reflected in the higher or lower costs of Query CD3, CD4, LS3, LS4, LS5 and LS7, as shown in Figure 6.6, specially LS5 where Gen-Greedy, Sum-Greedy and CostFed each have their wins in minimising costs with different pricing functions settings.

To amplify how sensitive Gen-Greedy and Sum-Greedy are to price variations, and whether being sensitive to price changes is helpful for minimising query costs or not, we conduct experiments with additional variation to the pricing functions of the relevant data sources. To showcase our findings without including too many results charts, we only choose Query LS5 for the price sensitivity analysis in this thesis.

Figure 6.14 shows the query plan of LS5 generated by CostFed. Obviously, there are 3 data sources relevant to Query LS5: *DrugBank*, *ChEBI* and *KEGG*, and the maximum amounts of data collected from them are 4,772, 34,055 and 34,260, respectively. To see how our algorithms react to the various costs of collecting data for triple patterns, especially when their costs rank in different orders, we choose per pricing functions to enumerate all the possible cost rankings of the collection of the above data by varying the p parameter in the pricing functions of those data sources. The 6 permutations of the 3 data sources and their pricing functions are listed in Table 6.2, where $SeleC(d)$ denotes the selectivity price a triple pattern (row) costs from data source d . For example, the ranking order No. 2 is $SeleC(DrugBank) < SeleC(KEGG) < SeleC(ChEBI)$, which is captured by applying the corresponding pricing functions: $0.1 \times 4,772 < 0.1 \times 34,260 < 0.2 \times 34,055$.

Figure 6.15 presents the costs of the query plans of Query LS5 generated by CostFed, Gen-Greedy and Sum-Greedy, with the 6 different pricing function settings in Table 6.2. When all the data sources charge for data access in the same way, CostFed and Sum-Greedy generate different query plans with equivalent costs which are both cheaper than the costs of Gen-Greedy's in Case 1. This is related to the join-selectivity among the

TABLE 6.2: cost ranking cases

No.	Cost Ranking	DrugBank	ChEBI	KEGG
1	$SeleC(DrugBank) < SeleC(ChEBI) < SeleC(KEGG)$	$f(0,0,0.1)$	$f(0,0,0.1)$	$f(0,0,0.1)$
2	$SeleC(DrugBank) < SeleC(KEGG) < SeleC(ChEBI)$	$f(0,0,0.1)$	$f(0,0,0.2)$	$f(0,0,0.1)$
3	$SeleC(ChEBI) < SeleC(DrugBank) < SeleC(KEGG)$	$f(0,0,3)$	$f(0,0,0.1)$	$f(0,0,3)$
4	$SeleC(ChEBI) < SeleC(KEGG) < SeleC(DrugBank)$	$f(0,0,6)$	$f(0,0,0.1)$	$f(0,0,0.2)$
5	$SeleC(KEGG) < SeleC(DrugBank) < SeleC(ChEBI)$	$f(0,0,3)$	$f(0,0,3)$	$f(0,0,0.1)$
6	$SeleC(KEGG) < SeleC(ChEBI) < SeleC(DrugBank)$	$f(0,0,6)$	$f(0,0,0.2)$	$f(0,0,0.1)$



(a) Triple Pattern Legend

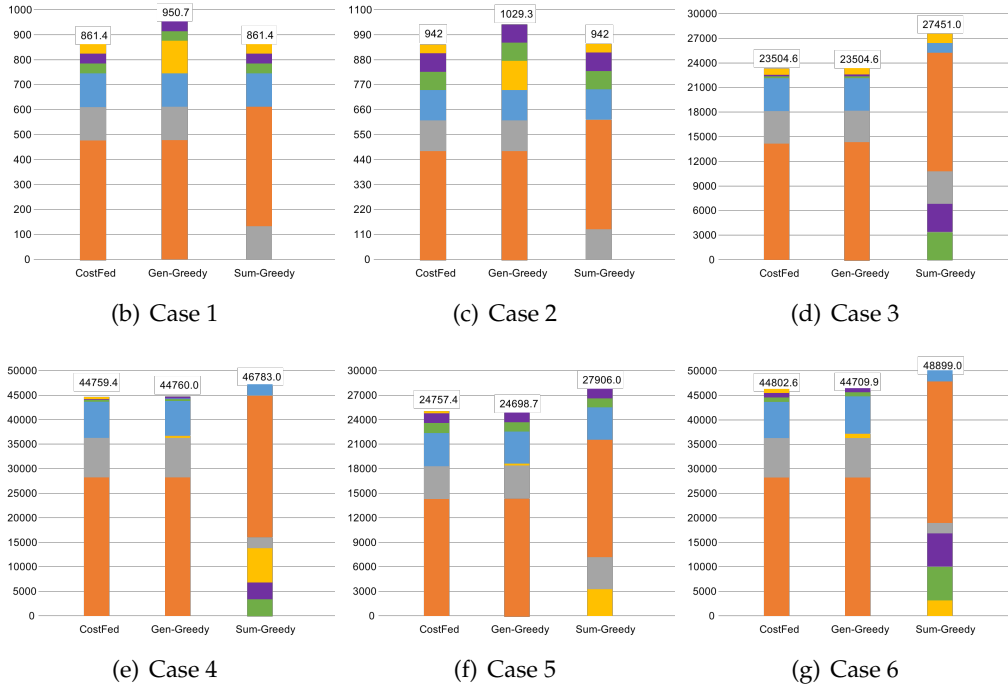


FIGURE 6.15: Query Plan Costs Generated by Gen-Greedy, Sum-Greedy and CostFed for Query LS5 in 6 Cost Ranking Cases

triple patterns of Query LS5, as we can see in Figure 6.14. With the sharp reduction of join-selectivity and lack of summary information, Gen-Greedy lost its efficiency while Sum-Greedy secured its advantage by adopting summary statistics in its estimation calculation.

When price variation weakens the cost differences caused by the selectivity of triple patterns, such as in Cases 4 and 6, where the selectivity cost from *DrugBank* is highest even though the selectivity from it is smallest, the profitability that buyers can gain with Sum-Greedy is not as clear-cut as in Case 1 compared to CostFed and Gen-Greedy. The reason is that when the cost differences fade, Sum-Greedy tends to arrange a query plan randomly. To be specific, the query order of *Dbpedia* and *ChEBI* is the essential part that causing the difference in the cost of query answers to Query LS5. When the

cost differences are smaller, such as in Case 3, the impact of random query planning on the performance of Sum-Greedy is more obvious. On the contrary, Sum-Greedy shows its advantage when the price difference among data sources is not so big as to erase the cost differences among triple patterns of queries, such as in Case 2. Moreover, as in the analysis above, Sum-Greedy can sense a better query plan than the other two algorithms when it is necessary to reconsider the order of the join-able triple patterns.

The faded cost difference is also precisely the reason for the outstanding performance of Gen-Greedy in Cases 3, 4, 5 and 6. Because Gen-Greedy lacks information about join-selectivity in its cost estimation calculation, when the price difference among the data sources becomes significant, the impact of the lack of join-selectivity information decreases as well, contributing to the better performance of Gen-Greedy. For example, in Cases 3 and 5, there is a thirty-fold price difference per result between *ChEBI* and *KEGG*, and Gen-Greedy manages to plan query execution in the cheapest way.

To summarise, both Gen-Greedy and Sum-Greedy are sensitive to price changes. The sensitivity to small price changes helps Sum-Greedy plan for executing queries more cheaply, but it turns out to be a shortage for Sum-Greedy for pursuing cheaper query plans when the price difference among the data sources elides the differences between the triple patterns. On the contrary, Gen-Greedy is more sensitive to price variations which is an advantage in generating query plans with lower costs.

6.3.2.4 Time

Our experiment is conducted over a hot cache, and every experiment for each query is executed 10 times, and then we exclude the biggest and smallest record to get the average value as the runtime of an algorithm. We average the running times of an algorithm under all 30 random pricing function settings of a query and plot them in Figure 6.16.

As would be expected, the extra calculation entailed in the effort to minimise costs increases the time taken for the local query engine to return results when it adopts the Gen-Greedy or Sum-Greedy algorithms in most cases. Nonetheless, Sum-Greedy performs as fast as CostFed in most cases. However, Gen-Greedy is surprisingly more time-efficient than CostFed and Sum-Greedy for LS3, LS5 and LS7. The specific reason behind this is that our server failed to obtain query results for some of the query plans generated by Gen-Greedy, as marked in Figure 6.6, whereas Sum-Greedy spends a long time collecting results. Such cases indicate that Gen-Greedy can be faster than CostFed but not Sum-Greedy. Overall, regarding the observation of the running time of all the experiments listed in Appendix B.4.2, Sum-Greedy is more time-efficient for price-based query planning.

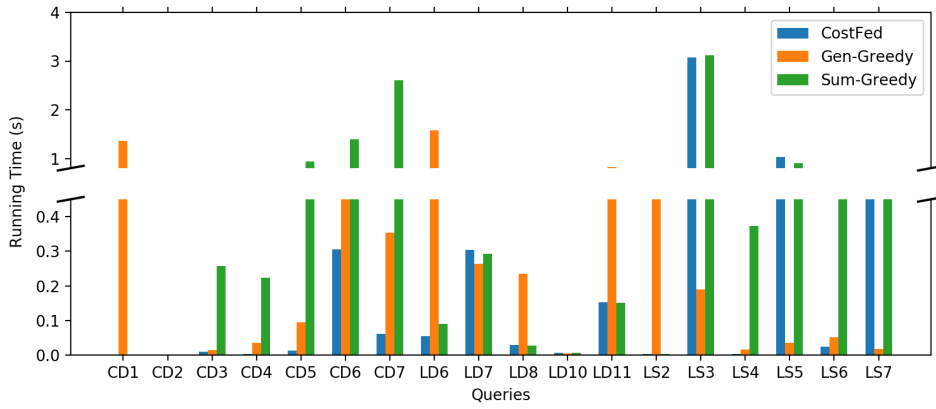


FIGURE 6.16: Average Running Time of the Three Algorithms for All Queries

6.4 Summary

To conclude this chapter, we address the following question: Can buyers find a cheaper way to purchase query answers by a local query engine in the Free Market? We first explain the problem with the example of John’s SPARQL query and the RDF data he can find, and then introduce our model for price-based query plans and cast the problem into finding a minimum spanning plan-tree over a complete query graph of a query with the cost of purchasing tuples matching a triple pattern presented as the weight of adding the triple pattern to the tree, where the cost is estimated by either general estimation or summary-based estimation. We then demonstrate two greedy algorithms applying the two estimation methods and evaluate their performance compared to the state-of-the-art query planning algorithm.

In general, the experiment results and subsequent analysis suggest that when buyers know nothing about data sources in the Free Market, or when the prices of data sources are obviously different, such as the last 3 cases in the experiments for LS5, Gen-Greedy would work better. If statistical summaries of the data sources are available, then CostFed and Sum-Greedy would be the first choice for buyers. Specifically, if there are more non-related data sources, i.e. non-join-able data sources, we would suggest clients to go with CostFed; otherwise, Sum-Greedy is a better choice to save money.

Chapter 7

Price-based Query Planning within Budgets in the Free Market

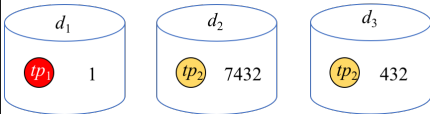
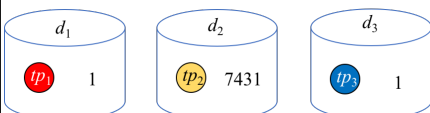
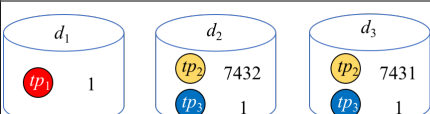
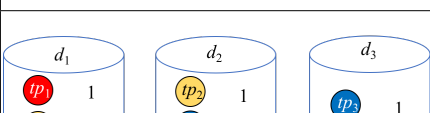
So far, this work has addressed the problem of planning query executions based on the prices of data for a local query engine in a free market. However, it is still possible that the query answer returned by executing a minimum cost query plan exceeds the limited budget of buyers. Furthermore, instead of ignoring the optimisation in respect to the time-efficiency of query plans, balancing the optimisation on runtime and costs of query plans will be of more benefit to buyers when the costs of query plans are within budgets.

This chapter investigates the problem of price-based query planning with a budget constraint. We first present a description of our research problem, formalise the problem based on the previous model of query plan-tree and cost estimation methods presented in Chapter 6 and demonstrate our approximate algorithm for solving the problem, followed by a summary of the work in this chapter.

7.1 Problem Description

Recall John's query and its selected sources in Chapter 6. We enumerate all the possible cases of data sources in a free market that John's local query engine may face while searching for answers to the query: (1) the market knows that the query has no answer; (2) the market does not know whether there exists an answer before execution; (3) no answer is returned within John's budget; (4) an answer is found within the budget. Table 7.1 illustrates the above cases with 4 free markets, A, B, C and D , where 3 data sources, d_1, d_2 and d_3 , are available in each market. The numbers beside the symbol of triple patterns in a data source represent the selectivity of a triple pattern in that data source. Note that there is only 1 triple that matches triple pattern tp_2 and is joinable

TABLE 7.1: Examples for Possible Query Results and Costs in Different Cases

Market	Relevant Data Sources of Triple Patterns	Results		Sunk Cost	
		Exist	Return	Minimum	Maximum
A		No	Empty	0	0
B		No	Empty	$c(d_3, tp_3)$	$c(d_1, tp_1) + c(d_2, tp_2)$
C		Yes	Empty	$c(d_3, tp_3)$	$c(d_1, tp_1) + c(d_2, tp_3) + c(d_3, tp_3) + c(d_3, tp_2)$
D		Yes	1	$c(d_1, tp_1)$ $+ \min\{c(d_2, tp_2), c(d_3, tp_2)\}$ $+ \min\{c(d_2, tp_3), c(d_3, tp_3)\}$	$c(d_1, tp_1) + c(d_2, tp_2)$ $+ c(d_3, tp_2) + c(d_2, tp_3)$ $+ c(d_3, tp_3)$

to tp_3 . When the selectivity of tp_2 in a data source is 7,431, it indicates that the above specific triple does not exist in the data source; otherwise, we mark the selectivity as 7,432 or 1 in a data source. The *Result* columns mark whether a market has answers to the query and the number of returned answers after the market executes the query. The *Sunk Cost* columns refer to the minimum and maximum costs that a market incurs when executing a query but returning no answers. Let $c(d, tp)$ denote the cost of purchasing triples matching triple pattern tp from data source d . Since the pricing functions of those data sources affect the cost of query answers, in order to analyse and compare the progress of a local query engine in the above markets, we assume that all the data sources price each triple at the same price. Hence, the cost of collecting answers to the query in the listed 4 markets are as follows.

- Market A: Since no triple matches tp_3 , no answer exists for the query. Before executing the query, the local query engine can return to John just after the process of source selection that no answer exists.
- Market B: Although there is also no answer in market B, since d_2 has no triples matching tp_2 to join with tp_3 , the local query engine is not confident to say this before executing the query, because all the triple patterns have matched triples. Thus, the cost of acknowledging that no answer exists is either as low as $c(d_3, tp_3)$ or as high as $c(d_1, tp_1) + c(d_2, tp_2)$.
- Market C: Both d_2 and d_3 have triples matching tp_2 and tp_3 , but there are no triples for tp_2 to join with tp_3 in d_3 . When the cost of accessing both d_2 and d_3 for triples matching tp_2 exceeds the budget, the local query engine would spend either $c(d_1, tp_1)$ or $c(d_1, tp_1) + c(d_2, tp_3) + c(d_3, tp_3)$ and tell John that his budget is

not enough for an answer. The worst case could be that the query engine pays for the access to d_3 and reaches the same conclusion.

- Market D: The very triples that contribute to the results of John's query exists. Considering the overlap of data sources, the local query engine finds a query plan to access data sources and collect results for the query within John's budget. The query plan avoids querying and paying both d_1 and d_2 for collecting the same triples matching tp_2 , and d_2 and d_3 for tp_3 . Therefore, it ensures that the cost of the returned results is no greater than the budget.

The goal of this chapter is to deal with the challenges in market C and D where, in fact, the answer to a query exists without exceeding the budget, but the local query engine needs to cut the purchase of unnecessary data while planning for the execution and improve the time-efficiency of query plans at the same time. We also try to reduce the sunk cost which exists in market B to the minimum. Note that this work will focus on the offline optimisation which means the optimisation process is taken in query planning instead of during the execution. Nonetheless, we will keep the online optimisation, which optimises a query plan while executing it based on real-time intermediate results, as the following work of this thesis.

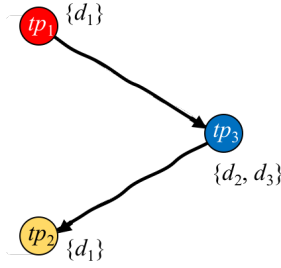
7.2 Theoretical Framing

We frame the problem of price-based query planning with a budget constraint by the model of a query plan-tree and cost estimation methods presented in Chapter 6. Instead of applying the estimated cost as the weight of incrementally connecting a node in a query graph to a tree, we let the weight represent the computation cost and treat the budget as an additional constraint for finding a spanning tree. Therefore, the problem is formalised into a weight-cost constraint minimum spanning tree problem. A solution to the problem requires minimum computation and costs no greater than the budget, implying a query plan with optimal time-efficiency as well as the satisfaction of the budget constraint.

7.2.1 Weight

The computation cost of joining a triple pattern to a tree can be illustrated by multiplying the cardinality of the matching terms of variables after a join. Therefore, we define the weight of adding a triple pattern tp_j to a tree, \mathcal{T} , via the edge $e < tp_i, tp_j >$ and tp_i existing in \mathcal{T} , as

$$w(e) = \prod_{i=1}^k \text{amount}(v_i, D'), \quad \text{amount}(v_i, D') \in \Pi(\mathcal{T} \bowtie tp_j) \quad (7.1)$$



(a) A Plan-Tree

$\mathcal{T}_0 = \emptyset$
$\mathcal{T}_1 = (N_1, E_1), N_1 = \{tp_1\}, E_1 = \emptyset,$ $EstimatedAmount(tp_1) = \langle amount(?party, \{d_1\}) = 1, amount(?x, \emptyset) = 0, amount(?page, \emptyset) = 0 \rangle,$ $\Pi(\mathcal{T}_0 \bowtie tp_1) = EstimatedAmount(tp_1)$ $= \langle amount(?party, \{d_1\}) = 1, amount(?x, \emptyset) = 0, amount(?page, \emptyset) = 0 \rangle$
$\mathcal{T}_2 = (N_2, E_2), N_2 = \{tp_1, tp_3\}, E_2 = \{e < tp_1, tp_3 >\},$ $EstimatedAmount(tp_3) = \langle amount(?party, \emptyset) = 0, amount(?x, \{d_2, d_3\}) = 2, amount(?page, \emptyset) = 0 \rangle,$ $\Pi(\mathcal{T}_1 \bowtie tp_3) = Join - Calculation(\mathcal{T}_1, EstimatedAmount(tp_3))$ $= \langle amount(?party, \{d_1\}) = 1, amount(?x, \{d_2, d_3\}) = 2, amount(?page, \emptyset) = 0 \rangle$
$\mathcal{T}_3 = (N_3, E_3), N_3 = \{tp_1, tp_3, tp_2\}, E_3 = \{e < tp_1, tp_3 >, e < tp_3, tp_2 >\},$ $EstimatedAmount(tp_2) = \langle amount(?party, \emptyset) = 0, amount(?x, \{d_1, d_2\}) = 14863,$ $amount(?page, \{d_1, d_2\}) = 14863 \rangle,$ $\Pi(\mathcal{T}_2 \bowtie tp_2) = Join - Calculation(\mathcal{T}_2, EstimatedAmount(tp_2))$ $= \langle amount(?party, \{d_1\}) = 1, amount(?x, \{d_2, d_3\}) = 2, amount(?page, \{d_2, d_3\}) = 2 \rangle$

(b) Incremental Progress of Adding Triple Pattern to a Plan-Tree

FIGURE 7.1: Process of Generating a Plan-Tree of John's Query in Market C

where $D' = \cup_{tp \in \tilde{N}} \mathcal{L}(tp) \cup \mathcal{L}(tp_j)$. The weight of \mathcal{T} is

$$W(\mathcal{T} = (\tilde{N}, \tilde{E})) = \sum_{e \in \tilde{E}} w(e) \quad (7.2)$$

where the calculation of the sum function follows the same order of incrementally adding triple patterns to a tree.

For instance, Figure 7.1 presents a query plan-tree of John's query to execute in market C and the weight of each process is listed in the subfigures. The weight of adding node tp_3 via edge $e_1 < tp_1, tp_3 >$ is:

$$w(e_1) = amount(?party, \{d_1\}) \times amount(?x, \{d_2, d_3\}) = 1,$$

which is a cross-domain join, and the following weight of edge $e_2 < tp_3, tp_2 >$ for \mathcal{T}_2 to span is:

$$w(e_2) = amount(?party, \{d_1\}) \times amount(?x, \{d_2, d_3\}) \times amount(?page, \{d_2, d_3\}) = 4.$$

7.2.2 Cost Constraint

Recall the cost estimation method in Chapter 6. The estimated cost of adding a node is $c(tp)$, where triple pattern tp is corresponding to the node. Hence, a cost constraint to represent a limited budget, B , for a query plan-tree $\mathcal{T} = (\tilde{N}, \tilde{E})$ is:

$$\sum_{tp \in \tilde{N}} c(tp) \leq B \quad (7.3)$$

where the calculation of the sum follows the same order of adding nodes.

For the example generating the plan-tree in Figure 7.1, the cost of incrementally adding node tp_1 , tp_2 and tp_3 are:

$$\begin{aligned} c(tp_1) &= PF[d_1](I(d_1)) = PF[d_1](amount(?party, \{d_1\})) = PF[d_1](1) \\ c(tp_3) &= PF[d_2](I(d_2)) + PF[d_3](I(d_3)) \\ &= PF[d_2](amount(?x, \{d_2\})) + PF[d_3](amount(?x, \{d_3\})) \\ &= PF[d_2](1) + PF[d_3](1) \\ c(tp_2) &= PF[d_2](I(d_2)) + PF[d_3](I(d_3)) \\ &= PF[d_2](amount(?x, \{d_2, d_3\}) \times amount(?page, \{d_2, d_3\})) + \\ &\quad PF[d_3](amount(?x, \{d_3\}) \times amount(?page, \{d_3\})) \\ &= PF[d_2](4) + PF[d_3](4) \end{aligned}$$

Therefore, given a SPARQL query Q and a constant budget B , we formalise the question of price-based query planning within a limited budget into a weight-cost constraint minimum spanning tree problem (CMST) (Ravi and Goemans) on a weighted graph by:

$$\begin{aligned} \mathcal{X} : \operatorname{argmin} W(\mathcal{T} = (\tilde{N}, \tilde{E})) &= \sum_{e \in \tilde{E}} w(e) \\ \text{s.t. } \sum_{tp \in \tilde{N}} c(tp) &\leq B \end{aligned} \quad (7.4)$$

which is a NP-hard problem and only has approximate solutions for now.

The weight $w(e)$ and cost $c(e)$ of an edge defined by Equation 7.2 and 7.3 are not constants but changes according to the order in which a tree spans a node and the sources chosen to be accessed for the corresponding triple pattern. This makes the greedy algorithm incapable of finding the minimum spanning trees. Nonetheless, the greedy algorithm is the foundation of all the existing solutions to both a simple MST problem and a CMST problem. Thus, instead of applying existing solutions to the CMST problem, we propose a pruning method and an approximation method based on Lagrangian relaxation (Lemar  chal, 2001) so as to approximate the optimal answers.

The pruning method is based on the minimum spanning tree algorithms. It adopts the spanning query plan-tree of a complete query graph with minimum costs and prunes the data sources to collect matched triples to triple patterns when the data sources make less contribution to the final query answer, until the estimated costs of a query plan are no greater than a budget. Lagrangian relaxation is a technique well suited for optimisation problems with constraints. It separates the constraints into two groups: the easy constraints and the hard constraints, and then eliminates the hard constraints from the constraint set by 'removing them to the objective function through a number of multipliers, to generate a relaxed version of the problem' (Şebnem Yılmaz Balaman, 2019). Therefore, the original problem is converted into a series of simplified problems with easy constraints, the solving of which approximates the optimal solution of the original problem at each step at the same time. We adopt the Lagrangian relaxation algorithm proposed by Ravi and Goemans, (2,1)-approximation algorithm, which always outputs a spanning tree with the minimum weight and the maximum 2 times the constraints, and then swaps edges in minimum spanning trees until finding a spanning tree that meets the constraints. Our approximation method reuses the relation process of Ravi and Goemans, but makes two adjustments in the swapping process, since the weights and costs of edges in a spanning query plan-tree are not constant. The following section illustrates the details of our algorithm design and implementation.

7.3 Minimum Runtime Plan-Tree within Budgets in the Free Market

To find the weight-cost constraint spanning tree as the price-based optimised query plan, we propose a minimum spanning tree planner, T-Planner. T-Planner is designed to generate query plans in two ways: (1) pruning the data sources of a minimum spanning plan-tree, which includes all the selected sources for each node, to reduce the cost of it until it meets the budget constraint, or (2) approximating the weight-cost constraint minimum spanning plan-tree by Lagrangian relaxation as Ravi and Goemans proved. In this section, we propose and implement a pruning algorithm based on the existing MST algorithms and an approximation algorithm for T-Planner.

7.3.1 Pruning Strategy

As discussed in Chapter 6, without the cost constraint that represents the budget limitation, the minimum spanning plan-tree of a query graph is the solution of problem \mathcal{X} . Hence, we adopt the foundation MST algorithm, greedy, to find the minimum spanning plan-tree for problem \mathcal{X} first, and then prune the relevant data sources with duplicated triples that match triples patterns to meet the cost constraints.

The target of the pruning process is to reduce the cost of the minimum spanning tree to satisfy the cost constraints while retaining as much of the completeness of query answers as possible. Therefore, our pruning approach is based on rejecting data sources that offer duplicated triples but make a smaller contributions in answering queries. We measure the contribution rate of a data source, C , by the extent to which the data source contributes to the answer to a query. Based on the join-selectivity in Section 6.2.2, we define the contribution rate C that a single data source d made according to the process of adding a tp_j to \mathcal{T} , is the ratio of the cost of intermediate results to the join-selectivity of triple pattern tp_j over d by

$$C = \frac{I^*(d)}{PF[d](I^*(d))} \quad (7.5)$$

where $I^*(d)$ is calculated by the equation in Equation 6.6.

Algorithm 4 shows our pruning approach. First, it calculates the cost of the given spanning tree (Lines 2-4). When that cost is greater than the budget (Line 5), following the order of adding triple patterns, it checks the selected data sources of each triple pattern (Line 7). The reason for this is that, without semantic information, we do not know whether rejecting a data source will affect the completeness of the query answer, and thus we only kick out data sources when the cost of a query plan exceeds the budget. If there is more than one data source (Line 8), the algorithm calculates the contribution rate of the data sources and saves it (Lines 9-12). Then, the algorithm sorts the data sources by the contribution rate and cuts the one with a lowest contribution rate (Lines 15-17). It recalculate the cost of the new spanning tree (Lines 18-21) and keeps kicking out data sources until the cost of the tree is no greater than the budget.

7.3.2 Approximation Strategy

Based on the (2,1)-approximation algorithm in [Ravi and Goemans](#), we first briefly introduce the basic idea of the algorithm while the supportive theory of which can be found in [Ravi and Goemans](#). Then, we set out our two adjustments in order to apply and implement them over weighted query graphs with a cost constraint.

7.3.2.1 Lagrangian Relaxation

Let $\gamma(e) \in \{0,1\}$ denote whether the edge e is in a spanning tree. Considering the budget constraint of problem \mathcal{X} in Equation 7.4 as the constraint, we can obtain a lower bound on the optimised value W by transforming it into the following simple minimum spanning tree problem:

$$f(z) = \min \sum_{e \in E} (w(e) + z \cdot c(e)) \cdot \gamma(e) - z \cdot B, \quad (7.6)$$

Algorithm 4: Kick(\mathcal{T}, B, D)**Input:** \mathcal{T} , a tree representing a query plan; B , the budget; D , the set of data sources.**Output:** \mathcal{T} , a tree representing a query plan within the budget.

```

1  $\sigma \leftarrow 0$ ;
2 foreach  $e$  in  $\mathcal{T}.E$  do
3    $\sigma \leftarrow \sigma + c(e)$ ;
4 end
5 while  $\sigma > B$  do
6   ConArray  $\leftarrow \emptyset$ ;
7   foreach  $tp \triangleright n$  in  $\mathcal{T}.N$  do
8     if  $\text{size}(\mathcal{L}(tp)) > 1$  then
9       foreach  $d$  in  $\mathcal{L}(tp)$  do
10         $c \leftarrow \mathbb{C}(d)$ ;
11        ConArray.add( $tp, d, c$ );
12      end
13    end
14  end
15  ConArray  $\leftarrow \text{sort}(\text{ConArray})$ ;
16   $tp, d, c \leftarrow \text{pop}(\text{ConArray})$ ;
17   $\mathcal{T}.N.\mathcal{L}(tp) \leftarrow \mathcal{L}(tp).\text{remove}(d)$ ;
18   $\sigma \leftarrow 0$ ;
19  foreach  $e$  in  $\mathcal{T}.E$  do
20     $\sigma \leftarrow \sigma + c(e)$ ;
21  end
22 end
23 return  $\mathcal{T}$ ;

```

with respect to the weight $w'(e) = w(e) + z \cdot c(e)$, for any $z \geq 0$. The value $f(z)$ is clearly a lower bound on W since any spanning tree which satisfies the above equation would give a no higher weight than in Equation 7.4. By maximising $f(z)$ over all $z \geq 0$, we can obtain the lower bound on W as:

$$LW = \max_{z \geq 0} f(z). \quad (7.7)$$

Let z^* denote the value of z which maximises $f(z)$, and $w^*(e) = w(e) + z^* \cdot c(e)$. To solve the Lagrangian relaxation and compute z^* , we calculate a minimum spanning tree with respect to $w'(e)$ for all values of z and two trees, \mathcal{T}_{min} and \mathcal{T}_{max} that have the smallest and largest cost. \mathcal{T}_{min} and \mathcal{T}_{max} can be obtained by applying the ordering $(w'(e), c(e)) < (w'(h), c(h))$ if $w'(e) < w'(h)$ or if $w'(e) = w'(h)$ and $c(e) < c(h)$. Let $C(\mathcal{T})$ denote the cost of a spanning tree \mathcal{T} . As [Ravi and Goemans](#) shows, z^* is the break point less than which $C(\mathcal{T}_{min}) > B$, and greater than which $C(\mathcal{T}_{max}) < B$. Hence, we can know $z < z^*$ if $C(\mathcal{T}_{min}) > B$, $z > z^*$ if $C(\mathcal{T}_{max}) < B$ or z is z^* otherwise. To find the minimum spanning tree at the value z^* without knowing z^* , we sort the edges

with respect to the weight $w^*(e) = w(e) + z^* \cdot c(e)$. For given two edges, e and h , we determine whether $w^*(e) < w^*(h)$, $w^*(e) = w^*(h)$ or $w^*(e) > w^*(h)$ by comparing the break point z_{eh} that $w^*(e) = w^*(h)$ with z^* since functions $w'(e)$ and $w'(h)$ are two linear functions. And the comparison of z_{eh} and z^* can be done by computing \mathcal{T}_{min} and \mathcal{T}_{max} at the value of z_{eh} .

Based on the theory in [Ravi and Goemans](#), there exists a spanning tree of minimum weight with respect to $w^*(e)$ where, at most, $LW \leq W$, and whose cost is less than $B + c_{max}(e)$ where $c_{max} = \max_{e \in E} c(e)$. To obtain the minimum spanning tree, let \mathcal{T}_{min}^* and \mathcal{T}_{max}^* denote the minimum spanning trees with smallest or largest cost at the value of z^* from all the minimum spanning trees that we get from the previous calculation, such that $C(\mathcal{T}_{min}^*) \leq B$ and $C(\mathcal{T}_{max}^*) \geq B$. There is a sequence of the minimum spanning tree $\mathcal{T}_{min}^* = \mathcal{T}_0^*, \mathcal{T}_1^*, \dots, \mathcal{T}_k^* = \mathcal{T}_{max}^*$ such that \mathcal{T}_i^* can be obtained from \mathcal{T}_{i+1}^* by an equal edge replacement ($0 \leq i \leq k$) following the analysis in [Wright \(1997\)](#) and simply return the first spanning tree whose cost is at most B . The sequence can be calculated by adapting Perrin's Algorithm ([Wright, 1997](#); [Martinez et al., 2017](#)) whose basic idea is to replace edges in a minimum spanning tree with the same weighted edges out of the spanning tree.

Given a graph G and a minimum spanning tree of G , \mathcal{T} , let h be an edge of G but not in \mathcal{T} , and $P(h, \mathcal{T})$ denote the path of h in \mathcal{T} that simply joins the vertices of h in \mathcal{T} . Considering a pair of edges of G , (e, h) , such that $e \in \mathcal{T}, h \notin \mathcal{T}$, e is on $P(h, \mathcal{T})$, and $w(e) = w(h)$, replacing e in \mathcal{T} will construct a new minimum spanning tree, $\tilde{\mathcal{T}} = (\mathcal{T} \setminus \{e\}) \cup \{h\}$. Then, (e, h) is an equal edge replacement in \mathcal{T} or (h, e) in $\tilde{\mathcal{T}}$. Repeatedly using the minimum cost edge h that is not in \mathcal{T}_{max}^* with its equivalent edges on $P(h, \mathcal{T}_{max}^*)$ will step by step construct the sequence we need.

7.3.2.2 Application and Implementation

Although the above analysis provides an approach to approximate the solution to problem \mathcal{X} , the weight and cost of an edge which is calculated based on the bind join of adding a new triple pattern to an intermediate tree are not constant but can change as the order of the join changes and as different data sources are accessed for the triple pattern. Hence, the ordering method and swapping maximum weight edges to get adjacent trees explained in Section 7.3.2.1 are invalid. Moreover, it is impossible to know the accurate cardinality of the join process before joining and thus the estimate weight and cost accurately.

Therefore, to adopt the algorithm, we need to find a method to calculate the weight of edges in or out of a query plan-tree and then compare their weights to find the edges with maximum weights and the edges with fewer weights. We propose two adjustments to apply the (2,1)-approximation algorithm to the problem of price-based query

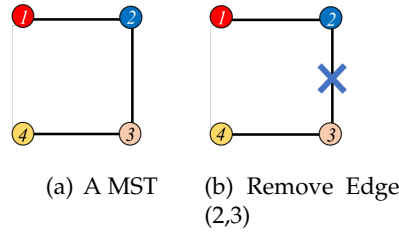


FIGURE 7.2: A Spanning Tree and An Edge Intended to Remove

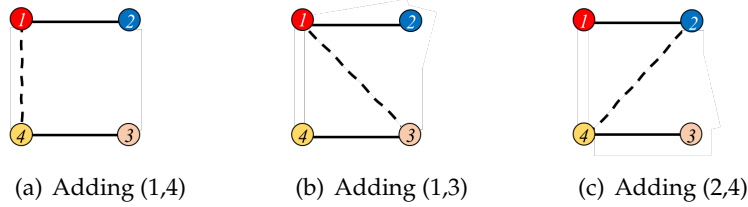


FIGURE 7.3: New Spanning Trees by Adding Edges

planning with budget constraints. First, \mathcal{T}_{min} and \mathcal{T}_{max} are calculated by applying a lexicographic ordering of edges instead of the ordering induced by w' . For example, when we are looking for \mathcal{T}_{min} , the ordering we use is $(w'(e), c(e)) < (w'(h), c(h))$ if $w'(e) < w'(h)$ or if $w'(e) = w'(h)$ and $c(e) < c(h)$. However, the weight and cost of adding a triple pattern also depends on which and how many data sources are accessed. Thus, we apply the minimum weight and cost of adding a triple pattern to find \mathcal{T}_{min} but the maximum weight and cost to find \mathcal{T}_{max} , and then implement the compare function of edges at the value z^* based on the calculation of \mathcal{T}_{min} and \mathcal{T}_{max} , as showed in Algorithm 6. The supporting theoretical analysis can be found in Appendixes C.1. The function, $MST()$, in lines 2-3 is the implementation of Perrin's Algorithm that deals with the computation of finding all the minimum trees at a value of z and then picks up \mathcal{T}_{min} and \mathcal{T}_{max} which have the lowest or highest cost.

The equivalent edges in the (2,1)-approximation algorithm has the same weight but different costs. Although a pair of equivalent edges of a query graph may have same weight, the weight of generating plan-tree keeps changing during the process of swapping edges and turning the tree into another minimum spanning plan-tree. Moreover, the weights of adding triple patterns also vary according to which and how many data sources are accessed for the triple pattern, since the budget may not cover constant weights which indicate that a query plan will access and pay for all the relevant data sources of triple patterns. Therefore, the replacing process of equivalent edges to generate plan-trees takes two phases: replacement and reduction. Replacement takes charge of the previously-explained edge swapping process, while reduction helps cut the weight and costs of an edge by removing a relevant data source.

As Algorithm 5 shows, the approximation strategy first generates the query graph of the input query, sorts all the edges of the graph by Algorithm 6 (Lines 1-2) and gets the

minimum spanning tree with maximum cost (Line 3). If the cost of the tree is greater than B (Line 4), repeat the equivalent edge replacement (Lines 5-9) which first sorts the edges again according to their costs from low to high (Line 5), pop up the edge that is not in the current tree and find its equivalent edges on its path in the current tree (Lines 6-7), pick the equivalent edge in the tree with maximum cost and replace it with the edge (Lines 8-9). Finally, Algorithm 5 returns a tree with weight at most and a cost less than B .

Algorithm 5: Approximation(BGP, B, D)

Input: BGP , the set of triple patterns; B , the budget;

D , the set of data sources.

Output: \mathcal{T} , a tree representing a query plan within the budget.

```

1  $G(N, E) \leftarrow QG(BGP)$ ;
2  $G.E \leftarrow \text{Sort}(G.E, \text{Compare}())$ ;
3  $\mathcal{T} \leftarrow \text{MST}(\text{"max"})$ ;
4 while  $C(\mathcal{T}_{min}) > B$  do
5    $G.E \leftarrow \text{Sort}(G.E, c())$ ;
6    $h \leftarrow \text{POP}(G.E \setminus \mathcal{T}.E')$ ;
7    $R(h) \leftarrow \text{EquivalentEdges}(h, P(h, \mathcal{T}))$ ;
8    $e \leftarrow \text{MaxCostEdge}(R(h))$ ;
9    $\mathcal{T} \leftarrow (\mathcal{T} \setminus \{e\}) \cup \{h\}$ ;
10 end
11 return  $\mathcal{T}$ ;

```

7.4 Evaluation

We evaluate the performance of the pruning and approximation algorithms in generating time-efficient query plans that cost no more than the budgets. We apply the same simulated free market in Chapter 6 for our experiments and set up budget constraints for each query.

In this section, we first explain the experiment settings and the metrics for the evaluation, then we present the experiment results and discuss the performance of the pruning and approximation algorithms.

7.4.0.1 Experiment Setting

The settings with regard to data sources, queries and the local query engine are the same as for the evaluation in Chapter 6. Distinctively, due to the impact of pricing functions and the purpose of observing the impact of different budgets on the performance of the proposed algorithms, we simplify the settings of the pricing functions for data sources to a price-per-triple function with the parameter p to be 0.02, and vary the

budget constraints for individual queries. We set the budget of each query to increase from 0.02 to as large as 1×10^{15} in order to cover all the cases from where the budget is not enough for a single answer to where the budget is greater than the cost of any answers.

7.4.0.2 Evaluation Metrics

To illustrate the performance of query planning in terms of efficiency in speed and budget satisfaction, we set the following three metrics:

- **Cost:** the cost of query answers collected by executing the query plans generated by algorithms.
- **Sunk Cost:** the price paid for a notification of an empty answer. The optimal goal of the algorithms is to obtain query answers within a budget constraint. However, when the algorithms failed in generating a query plan which costs greater than a budget in execution and received empty results, minimising the sunk cost paid for the execution is also their target.

We were also expecting to compare the completeness of the query answers in case the algorithms lose some answers when they try to meet the budget constraints. However, through our experiment, we noticed that reducing the number of data sources to access for the queries in FedBench (Schmidt et al., 2011) will only have two results: no answer or complete answers. For this reason, we will not present or compare the completeness in this work, but we plan to introduce a benchmark for the comparison of price-based query planning problem as the following work of this thesis.

7.4.1 Experiment Results

Note that both the prune and approximation algorithms do not guarantee the optimisation of their generated query plans but approximate the optimal solutions. To mark the best and worst query plans, we first enumerate all the query plans, calculate their costs under the settings of pricing functions of data sources, and then list their minimum and maximum costs at each different setting of budget constraints. Regarding the analysis of query plan enumeration in Appendix C.2 and the execution of all of the query plans for each setting of pricing functions, it would take an enormous amount of time to examine the performance of the prune and approximation algorithms for all the queries. For this reason, we pick Query CD6 and focus on conducting experiments with a varying budget over fixed pricing functions.

TABLE 7.2: Costs of Query Plans of Query CD6 at Different Budgets

Budget	Minimum		Prune		Approximation		Maximum	
	Cost	Result	Cost	Result	Cost	Result	Cost	Result
149750	149729.25	11	-	-	-	-	149729.25	11
149800	149729.25	11	149640.54	0	149757.74	0	149775.5	11
149850	149729.25	11	149822.64	0	149822.62	0	49846.47	11
149900	149729.25	11	149892.72	11	149892.72	11	149892.72	11
149950	149729.25	11	149928.31	11	149928.31	11	149928.31	11
150000	149729.25	11	149928.31	11	149928.31	11	149928.31	11

7.4.1.1 Costs and Sunk Costs

In Table 7.2, we list the costs of query plans generated by the prune and approximation algorithms and the number of results after executing them, along with the minimum and maximum cost from executing the enumeration query plans. The results of other settings of budgets are not shown in Table 7.2, since either smaller or greater budgets than the budgets listed in Table 7.2 lead to the same results as the ones already listed there.

With a budget within 149,750, both the pruning and approximation algorithm failed to find a query plan, even though there are query plans that have varied costs, as the minimum and maximum columns indicates. When the budget increased from 149,750 to 149,850, they managed to find a result with estimated costs lower than the budget. However, the query plans failed to collect any query results and led to their sunk costs. Notably, the prune algorithm causes a smaller sunk cost than the approximate algorithm. When the budget reached 149,900, both the pruning and the approximation algorithms succeeded in collecting query answers at the same costs, which are the maximum cost of all the plans.

To understand how the sunk costs are spent by the prune and approximation algorithms, we illustrate their query plans in Figures 7.4 and 7.5. The table on the left-hand side lists the triple pattern with a source, the cardinality of matched triples from the source and the costs. The tree on the right-hand side represents the query plan generated by the algorithm, and the symbols of data sources with a black delete line imply that the sources will not be accessed according to the query plan. The red arrow marks the stopping point of the execution of the query plan.

In the figures, we can see that the reason for the difference between the sunk costs of the pruning algorithm and the approximation algorithm is the starting triple pattern in their query plan. This is expected since the pruning algorithm, which is based on a plan-tree generated by greedy algorithms, spends less for each triple pattern while the approximation algorithm adjusts the plan-tree for the cost of the complete tree. However, due to the method of varying the setting of budgets, the experiment results do

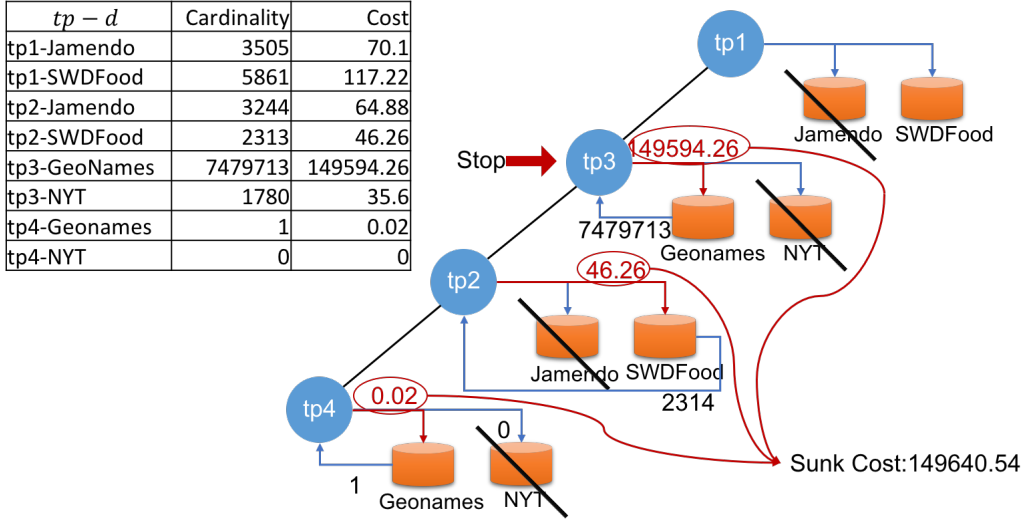


FIGURE 7.4: Query Plan for Query CD6 by the Prune Algorithm

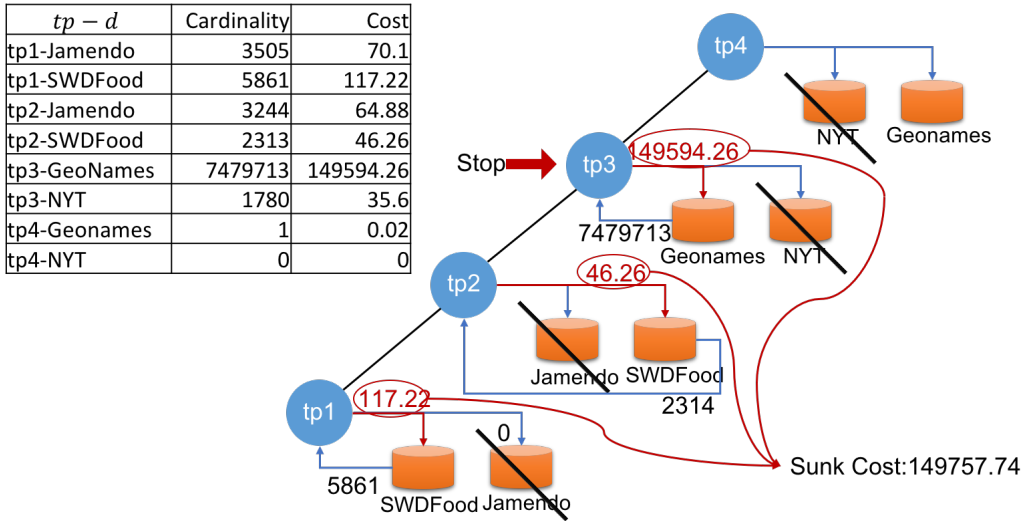


FIGURE 7.5: Query Plan for Query CD6 by the Approximation Algorithm

not present the advantage of the approximation algorithm in respect to the consideration of the cost of the complete tree. For instance, when the budget slightly increases by 1 from 149,800, the query plan in Figure 7.5 will return the query answer with the execution of triple pattern tp_4 over data source *Geonames*, while the query plan of the pruning algorithm still collects no results with a sunk cost of 149,640.54.

7.5 Summary

In summary, it has been shown in this chapter that the problem of price-based query planning with a budget constraint is challenging. We explain the problem following the example from the previous chapter, frame a weight query graph based on the

model in previous chapter and cast the problem into a weight-cost constraint minimum spanning tree problem. For solutions, we demonstrate our two ideas - the prune and approximation algorithms - to approximate the optimal solution to the problem and briefly evaluate their performance for Query CD6.

The results showcase that the prune algorithm has advantages in reducing the sunk cost because it is based on the MST generated by a greedy algorithm which tries to span each node at a local minimum cost. On the contrary, when the budget is approaching the limit of purchasing query answers, the approximation algorithm performs better in reducing the cost of the complete plan and returning query answers. However, both algorithms have a drawback that, compared to the optimal query plan, they do not succeed in reducing the cost of query answers.

Chapter 8

Market Comparison

Both the Data Emporium and the Free Market form a similar structure with respect to the general centralised markets for future data business, and yet offer different services to their clients, sellers and buyers, and deposit different tasks to them, as mentioned in Chapter 4 and 5. This chapter will contrast them and summarise the extra flexibility that the Free Market offers. First, we describe the framework of the existing markets, the Data Emporium and the Free Market, and then explain the similarities and differences in terms of their services for sellers and buyers.

8.1 Structure of Markets

The three subjects in the present examination are the general markets, the Data Emporium and the Free Market. The general markets refer to the markets mentioned in Chapter 2, whose major architecture shares common ground. Figure 8.1 shows the

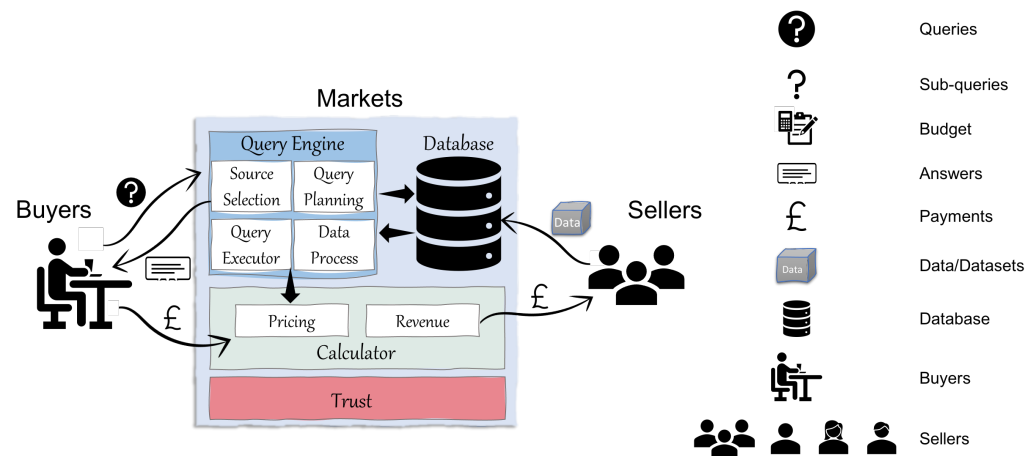


FIGURE 8.1: Structure of a General Data Market

FIGURE 8.2: Legends of Market Structures

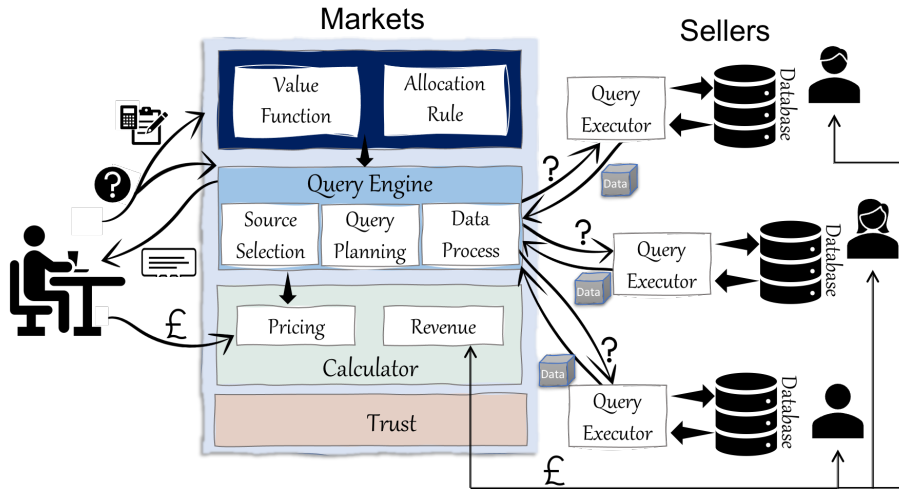


FIGURE 8.3: Structure of the Data Emporium

structure of the general markets and their components with the arrows illustrating the interactions among the components. A general market comprises four components: a database, a query engine, a calculator and the trust. The database stores the data sources of sellers. The query engine receives queries from buyers and then executes them on the database for query answers in a sequential process of source selection, query planning, execution and data processes; then, the query engine calls the calculator, which takes the query answers and/or the queries as input and outputs the price for buyers to pay. If the buyer agrees to pay the price, the market ends the deal after receiving the payment from buyers, and then calls the calculator to calculate the revenue share of each seller. As with any market (not necessarily a data one), trust among the participants is expected to be established in advance (Cioroica et al., 2019). The methods to form such a trust relationship are not within the scope of this work.

The structure of the Data Emporium is shown in Figure 8.3. With respect to general markets, the differences are the following:

1. Sellers are the host of the database and query executor and claim revenue to the market as their share;
2. Buyers can submit their budget constraints with their queries and configure value functions and allocation rules to the query engine;
3. Data Emporium will consider the cost, quality and utility of query answers based on the above settings during the searching process;
4. The required trust in the Data Emporium is slightly weaker than the general markets, since sellers are not expected to deposit trust in the marketplace about fair data prices but remain autonomous in pricing data; in the meantime, buyers can control the purchased data by setting up budget constraints and preferences instead of relying on the marketplace for the best answers to their requirements.

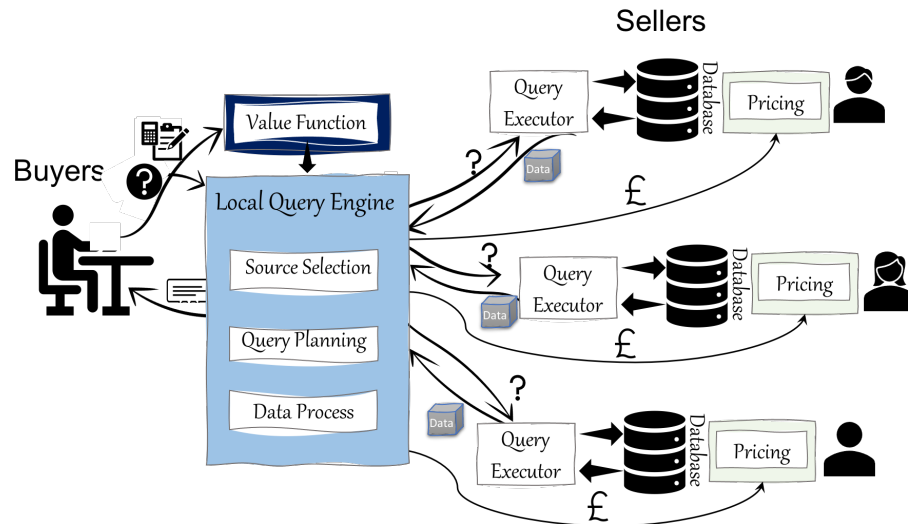


FIGURE 8.4: Structure of the Free Market

Figure 8.4 presents the structure of the free market and how buyers and sellers achieve deals on it. Buyers have a local query engine assisting them to search for query answers among sellers in the market. The engine receives as input a budget and a value function from the requirement of a buyer, and then selects the available sources for the buyers based on the information published in the market. It is the query planning component that takes care of the cost, quality and utility of query answers when it sends out sub-queries for buyers. Moreover, the Free Market has two significant differences with respect to the Data Emporium and the general markets. First, sellers have their own calculator components for pricing query answers as they prefer. They remain in complete control of their business. Secondly, buyers directly send out the payment to a seller after receiving the returned data. The settlement of business is easier and transparent to sellers.

8.2 Services for Sellers

We review the services of markets in respect to 4 key considerations for sellers: data management, trust, pricing and dependency, as detailed in Table 8.1.

8.2.1 Management

As a product, data requires good management. The data management oversees the storage and interaction of data. Storage indicates the place and mechanisms to locate data. From the engineering point of view, this can be as big as a globally-distributed storage system or as small as a single server at a specific place. Interaction defines where and how users access the data, i.e. the query interface and the query execution.

TABLE 8.1: Market Comparison of Services for Sellers

	General Market	Data Emporium	Free Market
Data Management: What data management services does a market offer?			
(1) Data Storage	Yes	No	No
(2) Query Interface	Yes	Yes	No
(2) Query Execution	Centralised	Federated	Decentralised
Trust: What authorization does a market require from a seller?			
(1) Accessing Data	Yes	Yes	No
(2) Assessing Quality	Yes	Yes	No
(3) Calculating Revenue	Yes	Yes	No
Pricing: How does a market suggest to calculate data prices?			
(1) Assigning Data Price	Yes/No	No	No
(2) Available Pricing Method	Per Data Set/View/Point	Per Data Item	Any
(3) Assigning Query Answer Price	Yes	Yes	No
Dependency: What is the relationship among sellers a market defines?			
	Collaboration	Collaboration	Collaboration/Solo

The query interface sets up the access point to the data and the types of requirements (SQL queries, SPARQL queries, etc.) that it responds to, while the execution refers to the strategy for executing queries over the data sources.

While general markets take care of all the above management work at a price for sellers, Data Emporium leaves sellers in charge of the storage but manages the query interfaces, ensures the collaboration of the data sources and federates the coming data requirements into sub-queries to the corresponding sellers for federated execution. In contrast, the Free Market aims to let sellers control all the management work. It leaves sellers the most space to decide their strategies to serve buyers based on their data, but requires a smart local query engine to assist buyers to purchase data in the meantime.

8.2.2 Trust

In current data trading, the ownership of data is secured by agreements. For example, if data users spread data without permission, they will end up with fines. However, trust is achieved through social policy but hard to build technically, since data are easy to copy, change and damage. In reality, sellers' first concern is the risk of giving free data access to a third party, since that would indicate a failure of their data control. A further trust issue is data quality assessment. When assessment of data quality is thoroughly up to others, sellers have no choice but to believe in the fair process. Thirdly, insecurity increases when sellers do not get the revenue they expected after serving data.

These trust concerns remain unsolved in both general markets and the Data Emporium. They ask for authorisation for free data access, assess data quality to make recommendations to buyers and are responsible for calculating the sellers' revenue shares. Specifically, both take the payment on behalf of sellers and calculate the revenue share of all sellers and transit it to sellers' accounts. However, since the Data Emporium can

be formed by a small union of sellers who choose to make deals together, it is more likely they already have trust in each other and sellers can claim their revenue shares. Distinctly and conversely, the Free Market does not get involved in the business between sellers and buyers and asks for no trust from sellers. The access, quality marking and charging for data are in the sellers' hands alone. Basically, sellers directly receive payments from buyers after they reach an agreement about the required data.

8.2.3 Pricing

Pricing is the strategy or function to calculate the price of a data requirement in a trading deal. Depending on the methods of pricing, sellers have varied control of the price of their data. In general markets, sellers assign the prices to data views or points. However, the price calculation of buyers' queries is under the authorisation of markets. Data Emporium assigns sellers with have full control of the data prices. Sellers set up strategies to charge the incoming requirements of their data, such as pricing data items, pricing queries, etc. In the meantime, the Data Emporium only sums up the bills from each seller to calculate the final price of the query answers. The Free Market makes a further step by giving up the domination in setting up the final price. Sellers have unlimited ways to assign data prices and notify buyers of the price of the answers to their requirements in the Free Market.

8.2.4 Dependency

Dependency describes the relationships among the sellers in a market. Any kind of entities owning data are potential sellers, e.g. independent sellers or large-scale marketplaces. To arrange answers to buyers' complex queries, sellers need to collaborate as general markets asks or to build a union and form a Data Emporium. Either being invited or autonomously collaborating, sellers depend on each other in general markets and the Data Emporium. However, the Free Market offers another choice for sellers: solo business, which indicates that each seller can simply trade their data alone without arranging agreements with other sellers or a marketplace.

8.3 Service for Buyers

To compare the service offered by these markets to buyers, our examination is to find their answers to 4 questions: (1) Does a market allow buyers to customise their financial constraints? (2) What service does a market offer buyers for their requirements? (3) What operations does a market undertake to collect data for buyers? (4) Will a market take the payment on behalf of sellers from buyers? The answers to these questions

TABLE 8.2: Market Comparison of Services for Buyers

	General Market	Data Emporium	Free Market
Budget: Does a market allow buyers to customize their financial constraints?			
(1) Configuration	None	Yes	Yes
(2) Variation	None	Yes	Yes
Efficiency: What service does a market offer buyers for their requirements?			
(1) Cost Reduction	None	Considered	Considered
(2) Double Charging	No/Considered	Considered	Considered
(3) Quality Control	No	Yes	Yes
(4) Utility	None	Considered	Considered
(5) Time	First Optimized	Second Optimized	Second Optimized
Searching Process: What operations does a market take to collect data for buyers?			
(1) Source Selection	Yes	Yes	No
(2) Query Planning	Yes	Yes	No
(3) Query Execution	Yes	No	No
(4) Data Process	Yes	Yes	No
Settlement: Will a market take the payment on behalf of sellers from buyers?			
	Yes	Yes	No

along with our comparison of their advantages and disadvantages, are listed in Table 8.2.

8.3.1 Budget

From the buyers' perspective, customising their data requirements with a budget limitation or a range is a financial expectation about markets. Nonetheless, not all data markets meet such an expectation. The general markets, for example, omit that service but focus on protecting sellers from arbitrage purchases. Conversely, buyers can configure their budget constraints in the Data Emporium, which is applied in the query execution while the Data Emporium improves the value of the returned query answers. The Free Market realises the above expectation from buyers but does not accept a budget constraint as input, since it does not manage query execution. This thesis offers price-based query planning methods to meet the buyers' expectation and improve the utility of data purchase for buyers as described in Chapter 6 and 7.

8.3.2 Efficiency

To compare the service quality of markets, we propose five assessment metrics for buyers, as listed in Table 8.2: cost reduction, double charging, quality control, utility and time.

- Cost reduction: describes the service of markets in minimising the price of the buyers' data requirements.

- Double charging: indicates the situation where buyers pay more than once for the same data, which needs to be avoided.
- Quality control represents the service of maximising the quality or value of the query answers.
- Utility: specifies the profit that the buyers can obtain from data purchases. Regarding the budget, utility means the service of maximising the value of query answers within budget constraints. Reducing the cost and avoiding double charging can help increase the utility.
- Time: measures the efficiency of a market in terms of answering buyers' queries.

Cost Reduction is not offered in the existing general markets which are more seller-friendly and focus on preventing arbitrage pricing. Neither does the service of quality control and utility. However, some markets avoid double charging by history-aware pricing or refunds [Upadhyaya et al. \(2016\)](#). Without considering the above services, runtime efficiency is the first and the only goal to optimise in general markets.

In contrast to general markets, both the Data Emporium and the Free Market consider cost reduction and double charging, control the quality of the query answers by a value function set up by buyers, and then further improve the utility for them. The Data Emporium accepts a value function and allocation rules from buyers. The value function defines the quality of data that buyers prefer, while the allocation rules explain how to purchase data at a cheaper price or without exceeding their budgets. Regarding the above two settings, the Data Emporium set maximising the utility as the first goal of query execution, which leaves the time as second priority for optimisation. The local query engine in the Free Market applies price-based query planning algorithms to reduce costs and try to avoid double charging. As in the Data Emporium, in the Free Market, it is up to buyers to set up the value function and budget for their queries. The local engine always sets the utility as the first optimising goal and pushes the time efficiency to the second.

8.3.3 Searching Process

There are 4 common searching processes to answer buyers' queries: source selection, query planning, query execution and data process. The first step, source selection, picks the data sources that are relevant to a query. This is followed by the second step of planning the targeted sources to which to send out the query or decomposed sub queries. Either the query or sub-queries will be executed in the third step, then the returned answers for the sources will be projected or selected in the data process step.

As we mentioned in Section 8.2.1, general markets offer centralised query execution, which infers that the searching process is made by the markets. The Data Emporium is slightly different. It selects sources and makes the query plans for buyers, and then

federates the query execution to the corresponding sources. After collecting answers from those sources, the Data Emporium will process the data based on the value function and the allocation rules if buyers have set it up as stated in Section 8.3.2. In respect of the decentralised query execution in the Free Market, all of the searching processes are the responsibility of buyers.

When the purchase decision is not made by buyers, they rely on the selected market to assist them to purchase data. This requires bold trust between buyers and the general market. In contrast, both the Data Emporium and the Free Market sidestep the trust challenge. The Data Emporium allows buyers to control part of the searching process by setting up value functions and allocation rules, whereas the Free Market leaves buyers to conduct the entire searching process with the assistance of a local query engine.

8.3.4 Settlement

The last process to establish a data purchase is to settle the payment. It is the market that takes the payment from buyers in the general markets and the Data Emporium. After that, both of them will share the revenue to sellers, as described in Section 8.2.2. In contrast, the Free Market steers clear of the settlement, so sellers receive payments directly from buyers. Although both types of settlement exist in the real world and are acceptable for sellers, the latter type offers more flexibility in finance and management for sellers.

8.4 Summary

This chapter has compared the designs of the general markets, the Data Emporium and the Free Market. To identify the similarities and differences, comparisons and contrasts of their structure and services are made. The relevance among them is clear in terms of providing places for data business. The Data Emporium and the Free Market diverge from the general markets on most services, however. Regarding the services for sellers, one of the obvious differences is that the Data Emporium and the Free Market drop the centralised data management, but suggest sellers store their data and execute queries. Although no significant divisions emerge between the Data Emporium and the general markets referring to trusts and dependency, the former is an autonomous unit of sellers and requires bolder trust and collaborations than the latter. The Free Market has some significant differences to them: it asks for no authorisation from sellers and does not get involved in the pricing; in addition, it offers another dependency option, being solo, to sellers. Sellers have maximum control of their business in the Free Market. In exchange, the sellers put more effort into the data management.

The comparison of the services for buyers shows that the Data Emporium and the Free Market take more account of buyers than the general markets. The most significant consideration is the budget and efficiency. The Data Emporium and the Free Market allow buyers to set up their budget constraints with their requirements and consider the profits of buyers in the business. There is no denying that such considerations increase the time of sending responses back to buyers. In addition, the buyers are more responsible for setting allocation rules and executing the local query engine in the Data Emporium and the Free Market, respectively. However, this additional waiting and effort may be considered cost effective when buyers have tight budgets and prefer valuable answers.

Chapter 9

Conclusion and Future Work

Data has become a critical asset for organisations to create value in optimising processes, developing business cases and implementing solutions. In the same way that raw materials are required to produce physical products, data needed to fuel digital products is also a product itself. Instead of investing money and labour into collecting data, exchanging and sharing data turn out to be a trend in monetising data and reducing the cost of collecting data, leading to the emergence of data markets as a kind of digital markets where data sellers and buyers trade data for money (Balazinska et al., 2011). Data Markets are the bridge that connects data buyers and sellers. The markets cooperate with sellers to monetise their data while supporting buyers to purchase data that meet their requirements. The existing studies of data markets offer good solutions to price data requirements in the form of queries, and allow data sellers to assign varied pricing strategies to their data sources. However, very limited research has been conducted on the technical problem of meeting the commercial requirements of sellers and buyers, which is trading data with an awareness of the diverse data pricing functions and the budget and utility constraints of purchases. These are a critical aspects in data trading since the different data prices, limited budgets and customised definition of utility are the essential factors that affect the decisions of sellers and buyers to trade data.

Previous studies either have not dealt with the purchase constraints, or tend to trade data based on trust, which implies sellers and buyers trust a marketplace for (1) fair transactions, namely that the marketplace will not set any sellers superior or inferior; (2) fair revenue sharing, whereby all sellers get paid for what they contribute; and (3) fair price, whereby buyers will get the best results for their money. However, such a marketplace-centred design is an impediment for both buyers and sellers with respect to their omitted expectations. From the perspective of sellers, pricing data or claiming revenue for their data contributions is a major aspect of their business. Although marketplaces bring various policies to protect sellers from arbitrage opportunities, as we described in Chapter 2, letting marketplaces price data and calculate their share does

not offer sellers enough flexibility in terms of managing data and controlling revenues. On the other hand, buyers expect marketplaces to make decisions for their benefit in the case that query answers can be derived from multiple sellers who offer duplicated data. If marketplaces do so, it will either end up with a decrease in the revenue of some sellers, or a decrease in the revenue of all sellers by splitting the payment equally. Both are against sellers' expectations of maximising revenue. Current markets tend to ignore the above expectations from buyers and the constraints they face, such as budget limitations and data preferences.

To make it easier for both buyers and sellers to trade data, this work has explored a method to remove the impediments and investigated the question of how to trade data with an awareness of the diverse data pricing functions autonomously set by sellers, and the budget and utility constraints for purchases set by buyers in two ways, and then has delivered a comparison of our methods with the existing models in terms of the satisfaction of buyers and sellers to their services. Our first attempt is to embed a component in the existing data markets to allow buyers to purchase data under their constraints while sellers can autonomously price their data. We meet the above-mentioned expectations of sellers by introducing a federated data market, Data Emporium, where the involvement of a marketplace only consists of processing queries and calculating their prices. Regarding the expectations of buyers, the challenge lies in finding the best purchase for their money with respect to their constraints of budgets and preferences when sellers apply access-dependent pricing functions (ADPF) to price the data requirements. This work formalises this problem in a federated data market, analyses the structure and complexity of the problem when ADPF is considered, and then presents a two-phase method to solve the problem. First, there is a compression phase aimed at minimising the price to pay for a set of data items. Then, the allocation phase looks for a subset of the complete answer which costs no greater than a budget with maximal utility. We have proposed a Cost Compression algorithm for the first phase that purchases subsets of the input tuple set according to the price of sources, instead of selecting sources offering the lowest priced data items and implement two heuristic algorithms, Greedy and 3DDP to approximate the optimal allocation while using the cost compression algorithm to minimise the cost of a subset of the complete answer based on the tuples that derive the answer.

This thesis has further explored the design of data markets to enable buyers to purchase data from multiple independent sellers (i.e. individual sellers or marketplaces) under their purchase constraints. We propose an architecture: Free Market, where buyers and sellers retain full autonomy in the data trade business. Sellers reserve the management of their data sources in terms of access control and pricing functions, while the marketplace in a Free Market only makes an announcement of the available data sources in the market. Buyers independently search for answers to their requirements and directly trade data with sellers via a local query engine. To satisfy a buyer's data requirement,

the local query engine is designed to work differently from the existing ones and faces the technical challenge of optimising query plans to meet the constraints of budgets and preferences/priorities in the requirement. We first model the above-mentioned challenge into a minimum spanning tree problem with the cost of data requests as the weight of the spanning trees. Then, we introduce our heuristic approaches to approximate the optimal query plan for a local query engine and evaluate their performance in comparison to the state-of-the-art query planning algorithm. After that, we investigate the price-based query planning with budget constraints, aiming to optimise the time efficiency of query plans without exceeding a limited budget, and demonstrate an approximate algorithm as a solution.

In the end, based on the review of related work, we deliver a comparison of our methods and a general adopted model in the existing data markets. The comparison explains their structure and services from the point of view of buyers and sellers, which is a brief guide for buyers and sellers when they make the first move to participate in data trades.

9.1 Future Work

The problems we address in this work are shown to be NP-hard. As such, an evaluation over a large scale of data sources and complicated queries is not appropriate. Instead, future work should look at realistic data sources and common queries to identify if any simplifying assumptions or optimisations can be made that will take the problem into PTime. In order to do this, the community needs a set of realistic benchmarks and data sources that include various settings of pricing functions, budgets and data utilities. Based on our experiment results, there are four factors that affect the efficiency and fairness of a data market benchmark: the structure of queries, the relevance between queries and data sources, the duplication rate among data sources and the pricing function of the data sources with duplicated data. Hence, to design a set of realistic benchmarks, we plan to (1) enumerate a list of triple patterns and the matching triples; (2) generate a list of queries based on the triple pattern list, where each query either has a different number of triple patterns, or have different ways to join triple patterns; (3) for each query, create a list of relevant data source sets of the query by putting the triples that match the triples patterns in the query into a set of data sources, where each set of data sources in the list consists of either a different number of data sources or a different rate of duplicated data; (4) for each set of data sources, enumerate all the possible ways to price their data based on the existing pricing functions. The generated queries and data sources are suitable to examine the efficiency and capability of a data market in serving different queries on top of various related data sources with different combinations of pricing functions.

A natural progression of this work is to explore a way to vary multiple parameters for a systematic comparison with other markets. We would like to further explore the sensitivity of Gen-Greedy and Sum-Greedy in a circumstance where multiple data sources are relevant to only one triple pattern on a series of queries which consist of the same triple patterns but in different orders, and the pricing functions of the involved data sources can vary in an enormous range of types and in the value of its parameters. Considerably more work can be done with a more suitable benchmark. For instance, we would suggest comparing the completeness of the query answers in a scenario that the algorithms lose some answers when they try to meet the budget constraints. In the follow-on work, we are aiming to try and pick out the triples matching triple patterns and allocate them to different data sources to create cases, where multiple data sources contribute to the complete query answers, and avoid the cases that a data source is relevant but its existence does not affect the completeness of the query answers. Moreover, with the created cases, we can explore the impact of various pricing function settings on the performance of the query planning algorithms.

Furthermore, we are aiming to explore how to recommend pricing functions to sellers, as well as the parameters according to their data and possibly based on the current queries as well as analyse the impact of the overlap rate among the data sources on the algorithms. Considering the different circumstances: (1) When a seller needs to collaborate with others to sell query answers to a buyer, if the seller changes the price of data, how does it affect the profits of the seller and the other collaborated sellers? (2) When a seller competes with other sellers when their data sources have overlaps, how do pricing function settings affect the profits with or without knowing the pricing functions of the competitors? We consider the marginalism and game theory as candidate models to optimise the profits of sellers when they change data prices. We aim to explore each circumstance, with a set of queries and budgets by setting the price of different data sources, i.e. sellers' data, to be equal and calculate the profits of sellers as the baseline, and then vary the data prices using a method to control the variation of parameters, i.e. change the price of sellers' data one by one or separate them into a collaborating group and a competing group and then change the price group by group.

The Free Market would be a fruitful area for further work. The efforts other than data, e.g. time and money, for buyers and sellers to put into trading data in the Free Market is an inspiring topic to discuss in the field of business. One of the interesting technical focuses is to find online/real-time query optimisation. We adapt cost estimations in this work to plan query execution without precise information about what can happen during execution. Applying intermediate information to adjust the query plans while executing a query is a possible method to improve the cost-efficiency of query optimisation, reduce the sunk costs and avoid empty answers. Specifically, an adaptive local query engine will execute a received query plan by sending out the first sub-query.

After receiving the intermediate results, it reconsiders and estimates the cost of the remaining sub-queries that are waiting to be sent out. The estimation results will help the query engine choose a sub-query to send out. The advantage of this model is that it uses more real information in estimation and, therefore, it may perform better in controlling the cost of query answers. On the contrary, the execution of a query in this model involves recalculation and estimation and, therefore, it requires more time to return query answers.

Appendix A

Purchase Allocation Problem

A.1 Purchase Allocation Problem Formulation

For formalization of an allocation to the problem, let $r(s) \in \{0, 1\}$ denote whether the solution s is in Λ or not. $R = \{r_1, r_2, \dots, r_k\}$ denote the allocation of solutions from S into Λ , where $r_i \in \{0, 1\}$ denotes whether the corresponding solution s_i is in Λ or not. Thus, the utility of an allocation is

$$\mu(\Lambda) = \sum_{s \in \Lambda} r(s) \times v(s). \quad (\text{A.1})$$

To allocate a solution s , buyers have to purchase the relevant triple set $RT(s)$. Let $z(t) \in \{0, 1\}$ represent whether a triple t is purchased or not. Thus, the above constraint is $r_s = 1 \iff \forall t \in RT(s), z(t) = 1$, i.e., $r_s = 1$ is no greater than the sum of all $z(t)$ of all triples in RT . We formalize this constraint into

$$|RT(s)| \times r(s) - \sum_{t \in RT(s)} z(t) \leq 0, \quad \forall s \in \Lambda. \quad (\text{A.2})$$

The same triple can exist in different sources with different prices (e.g., t_3 in Table 4.1(a)), buyers should pay only once for it, even if it's used for different solutions. To calculate the cost of purchased triples while data sources apply different pricing functions, PA has to ask for the price of triples based on the data sources where it purchased them. Therefore, PA has to track from what source each triple has been purchased. To do so, let a (0,1)-matrix $DT = (a_{ij})_{m \times n}$, where m is the amount of data sources in D and n equals the cardinality of triples in $RTA(\Lambda)$, represent the purchase of the triples in $RTA(\Lambda)$ over D . The binary variable $a_{ij} \in \{0, 1\}$ denotes whether PA purchases the j th triple t_j in $RTA(\Lambda)$ from data source d_i ($a_{ij} = 1$) or not ($a_{ij} = 0$). In DT , the j th column $COL_j = (a_{1j}, a_{2j}, \dots, a_{mj})$ represents the purchase of t_j over D^M , and the i th row $ROW_i = (a_{i1}, a_{i2}, \dots, a_{in})$ denotes the triples purchased from data source d_i . Thus, the

set of triples T_i purchased from d_i is

$$T_i = \{t_j | t_j \times p_{ij} \neq 0, t_j \in RTQ, p_{ij} \in Row_i, j = 1 \text{ to } N\} \quad (A.3)$$

Thus, for a query answer Λ with the purchase of the relevant triple set RTA , can be tracked by a DT that satisfies the following constraints:

Thus, we can formalize the pay-only-once constraint as

$$\sum_{i=1}^m a_{ij} \leq 1, \quad j = 1, \dots, n. \quad (A.4)$$

And $C(d_i)$ representing the cost of purchasing $RTA(\Lambda)$ from data source d_i is

$$C(d_i) = PF[d_i](\{t_j | a_{ij} = 1, j = 1, \dots, n\}).$$

Then, the cost of Λ is $c_\Lambda = \sum_{i=1}^m C(d_i)$ and the budget constraint turns into

$$\sum_{i=1}^m PF[d_i](\{t_j | a_{ij} = 1, j = 1, \dots, n\}) \leq B. \quad (A.5)$$

Summing up all the above constraints, an optimal allocation should satisfy the following problem:

$$\begin{aligned} & Y : \arg \max_{\Lambda \subseteq S} \mu(\Lambda) \\ \text{s.t. } & |RT(s)| \times r(s) - \sum_{t \in RT(s)} z(t) \leq 0, \quad \forall s \in \Lambda \\ & \sum_{i=1}^m a_{ij} \leq 1, \quad j = 1, \dots, n \\ & \sum_{i=1}^m pf(\{t_j | a_{ij} = 1, j = 1, \dots, n\}, d_i) \leq B \\ & a_{ij} \in \{0, 1\}, \quad i = 1, \dots, m, \quad j = 1, \dots, m \end{aligned} \quad (A.6)$$

A.2 Solution Size and Price Rate under Equal Utility Setting

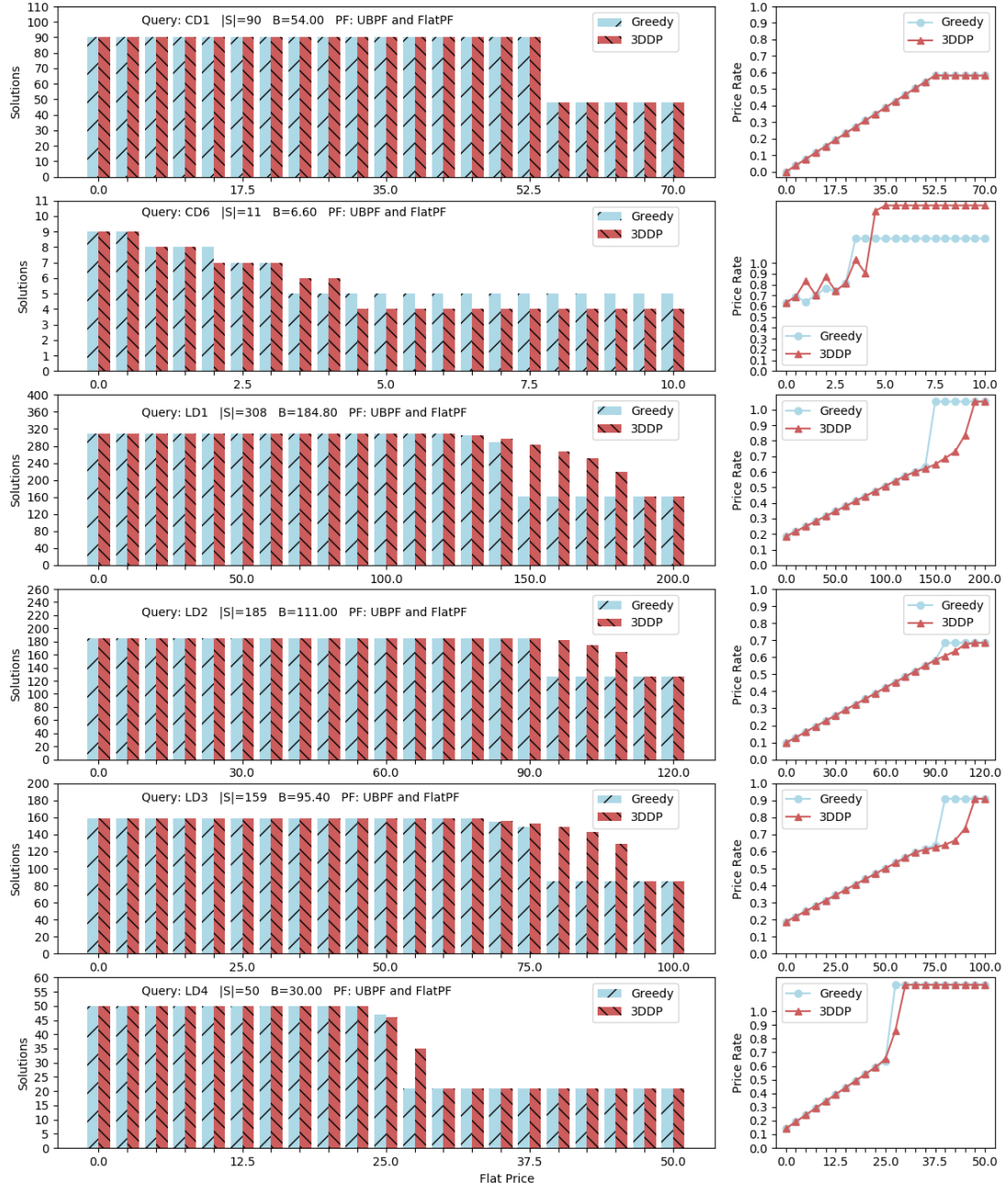


FIGURE A.1: Total Solutions in the Returned Allocations and Price Rate of Greedy and 3DDP in Settings of UBPF + Flat (1)

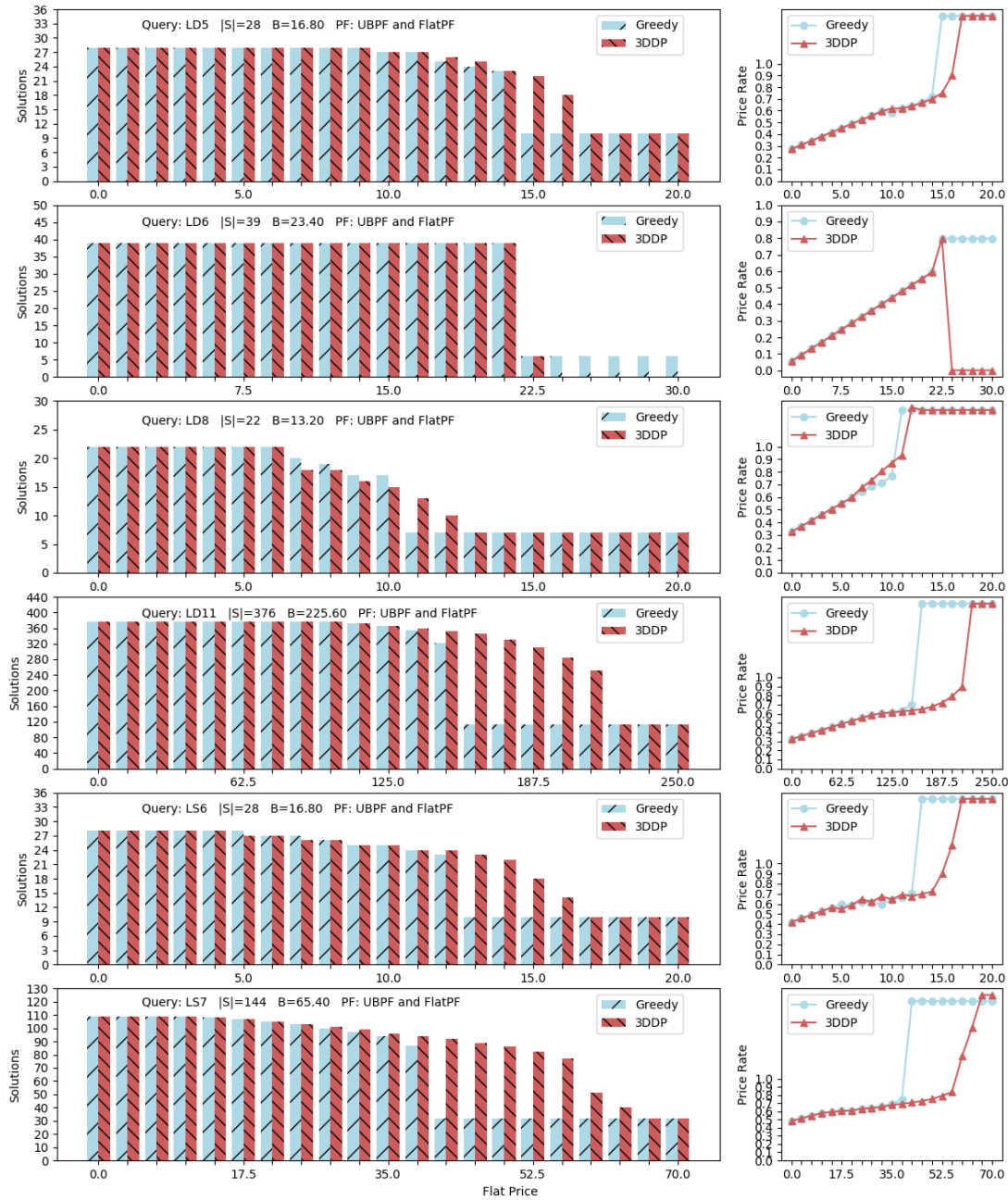


FIGURE A.2: Total Solutions in the Returned Allocations and Price Rate of Greedy and 3DDP in Settings of UBPF + Flat (2)

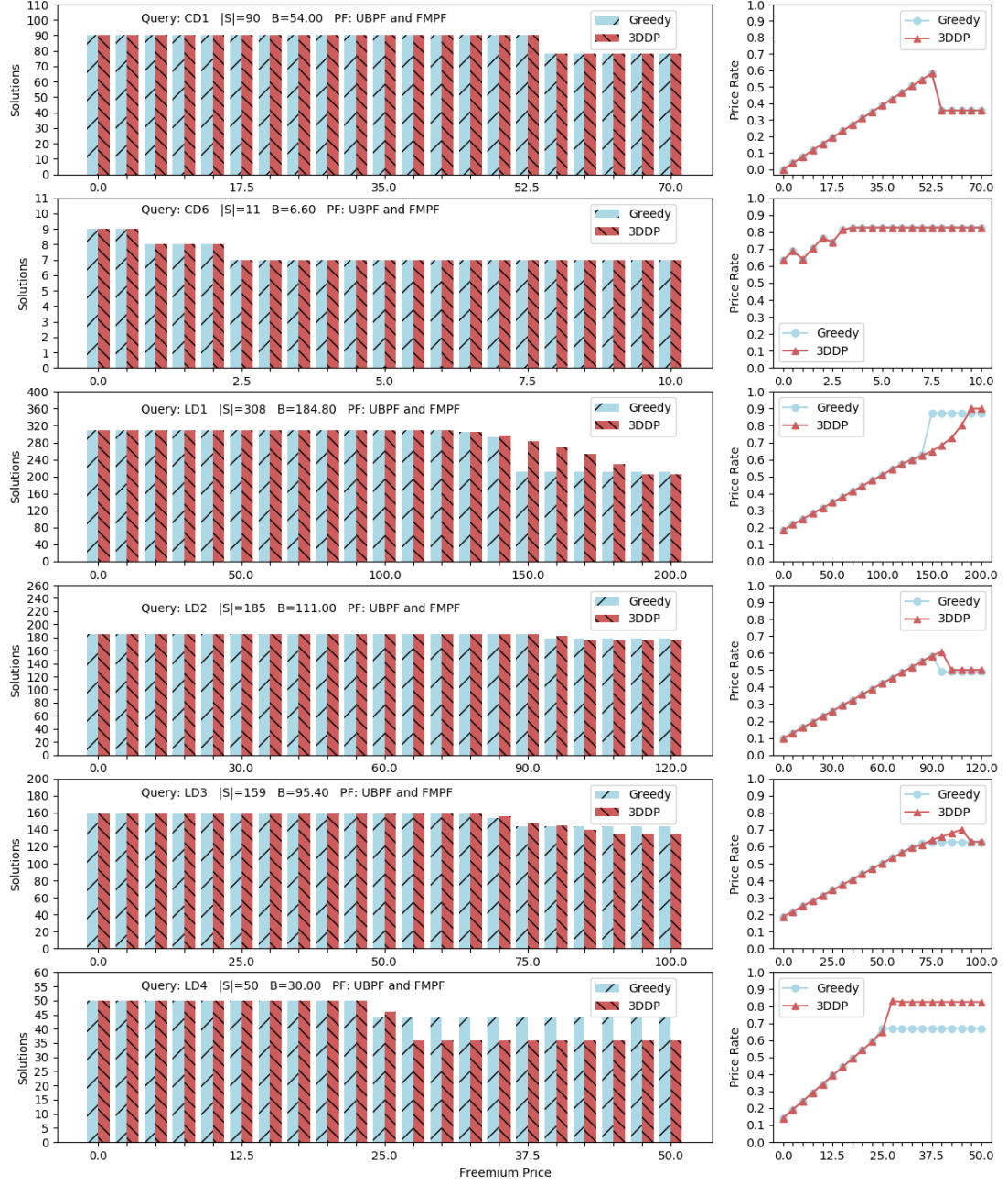


FIGURE A.3: Total Solutions in the Returned Allocations and Price Rate of Greedy and 3DDP in Settings of UBPF + Freemium (1)

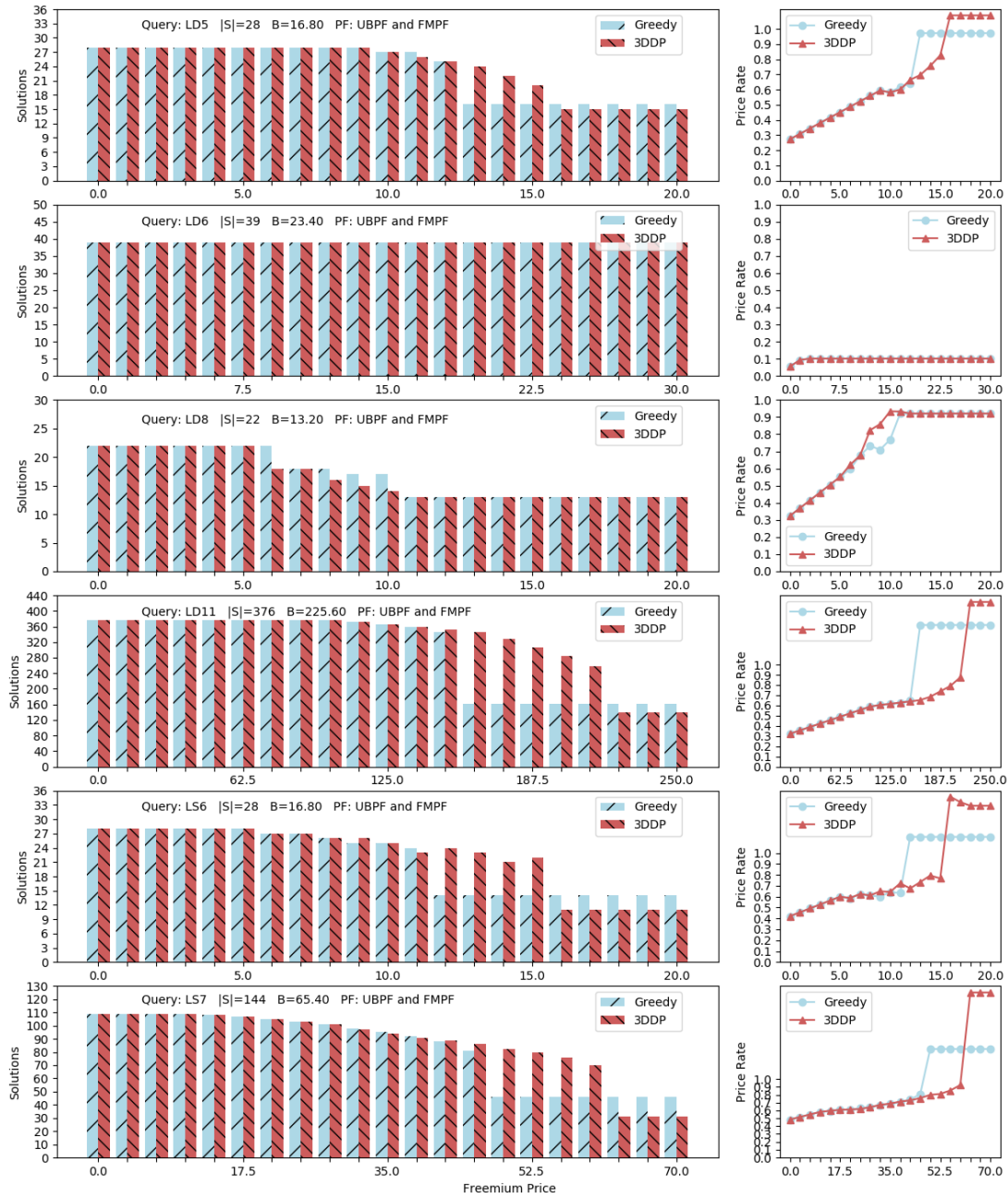


FIGURE A.4: Total Solutions in the Returned Allocations and Price Rate of Greedy and 3DDP in Settings of UBPF + Freemium (2)

A.3 Utility of Allocations under Sized-Distributions of Utilities Settings

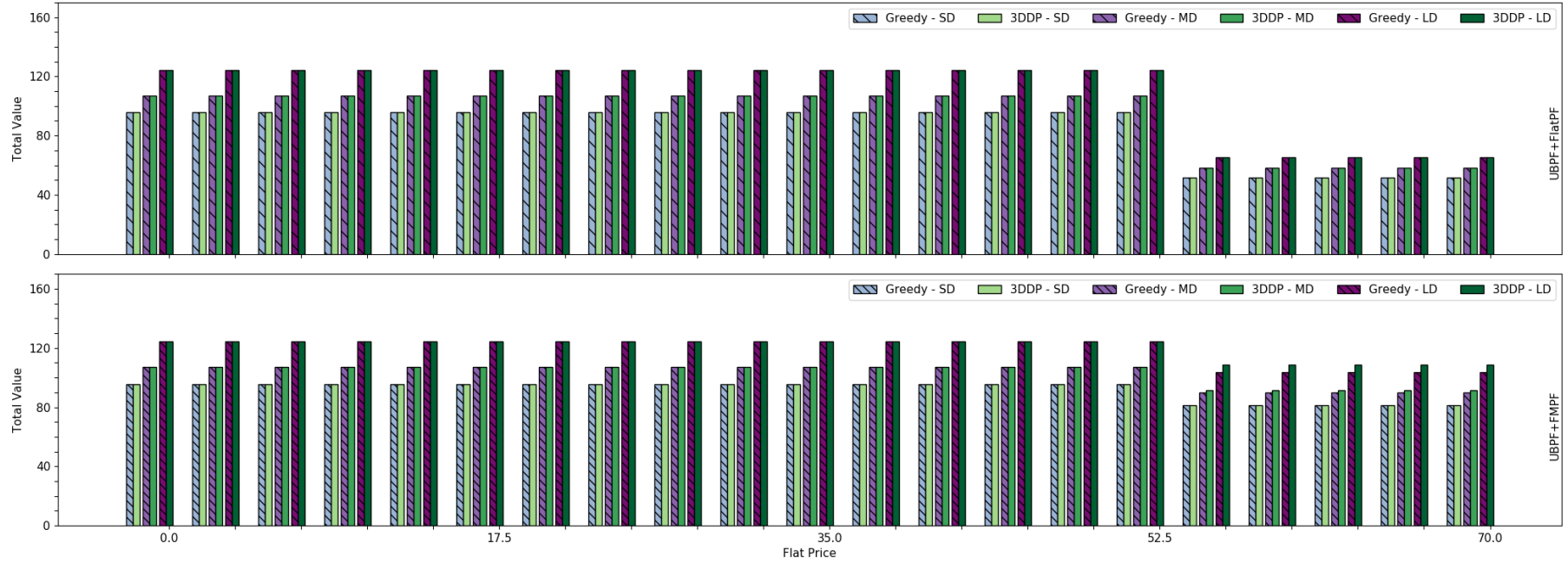


FIGURE A.5: Total Utility of the Allocations Returned by Greedy and 3DDP for Query CD1 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities

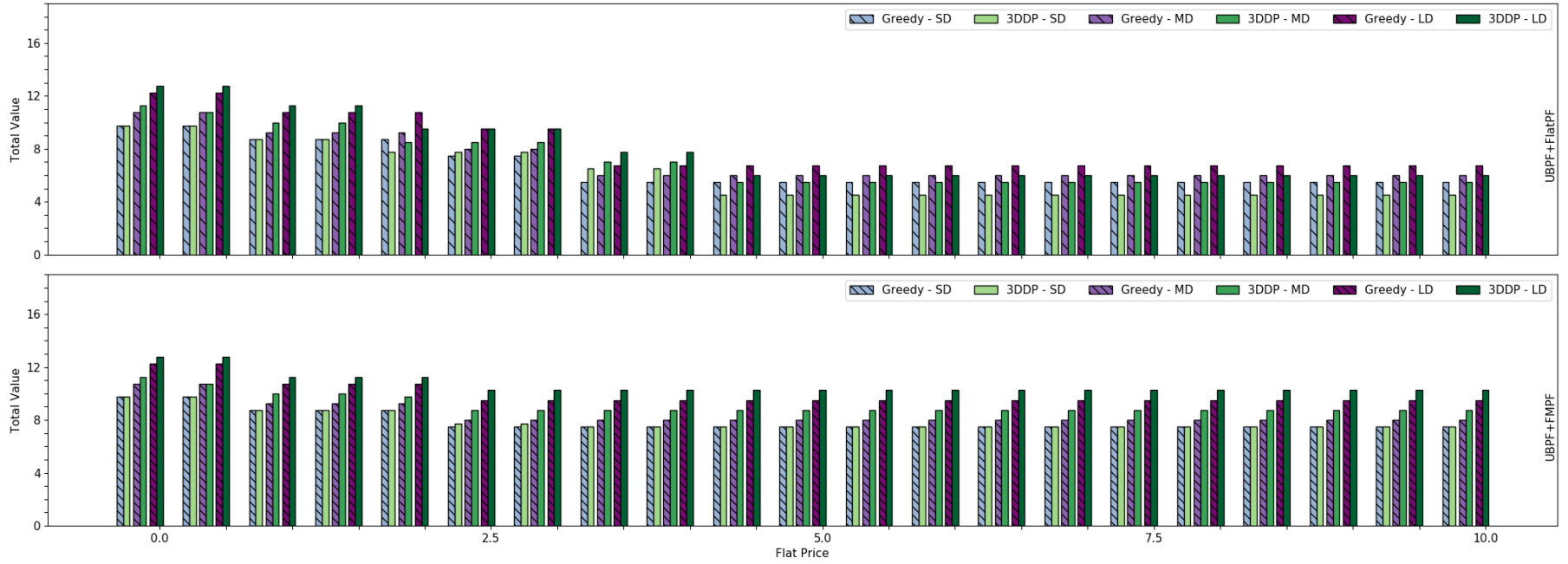


FIGURE A.6: Total Utility of the Allocations Returned by Greedy and 3DDP for Query CD6 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities

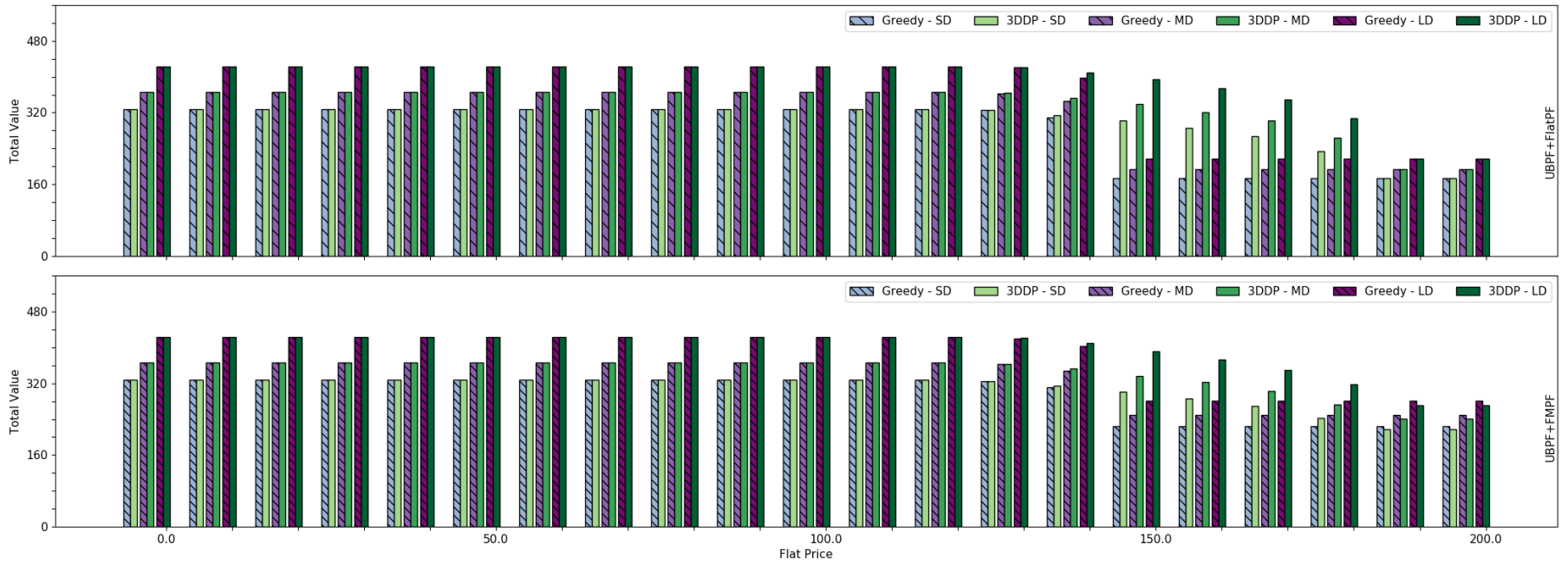


FIGURE A.7: Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD1 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities

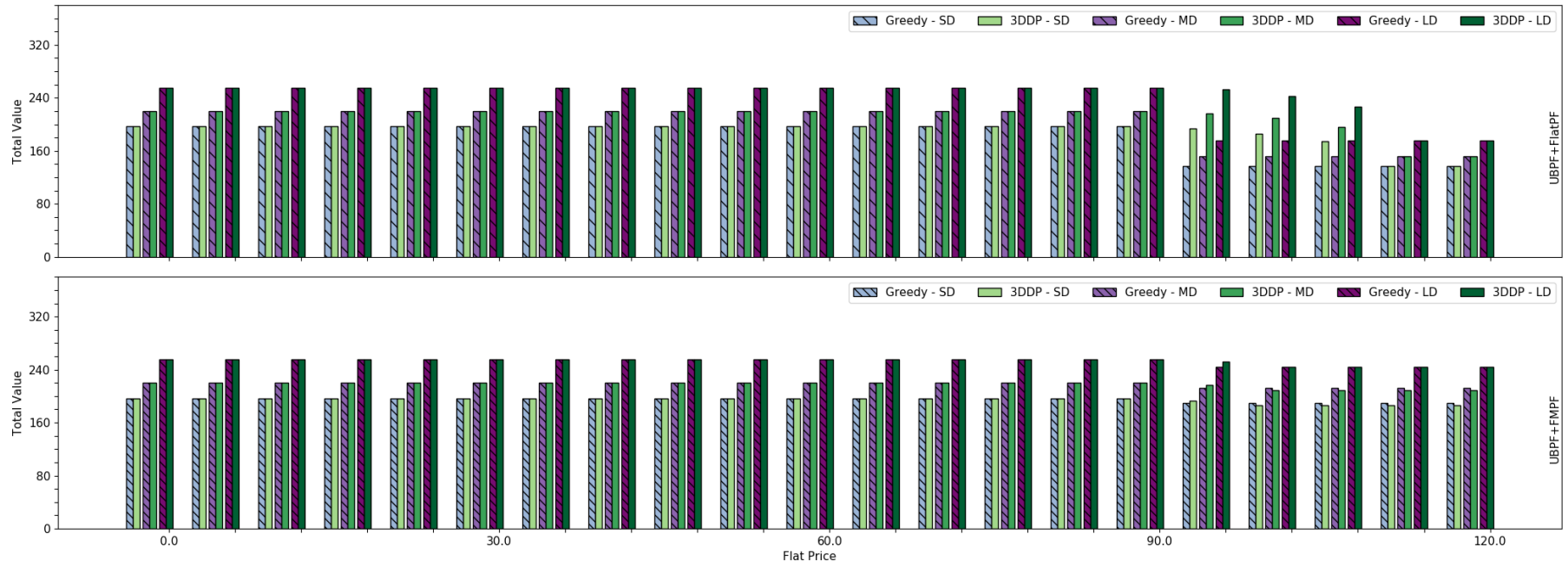


FIGURE A.8: Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD2 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities

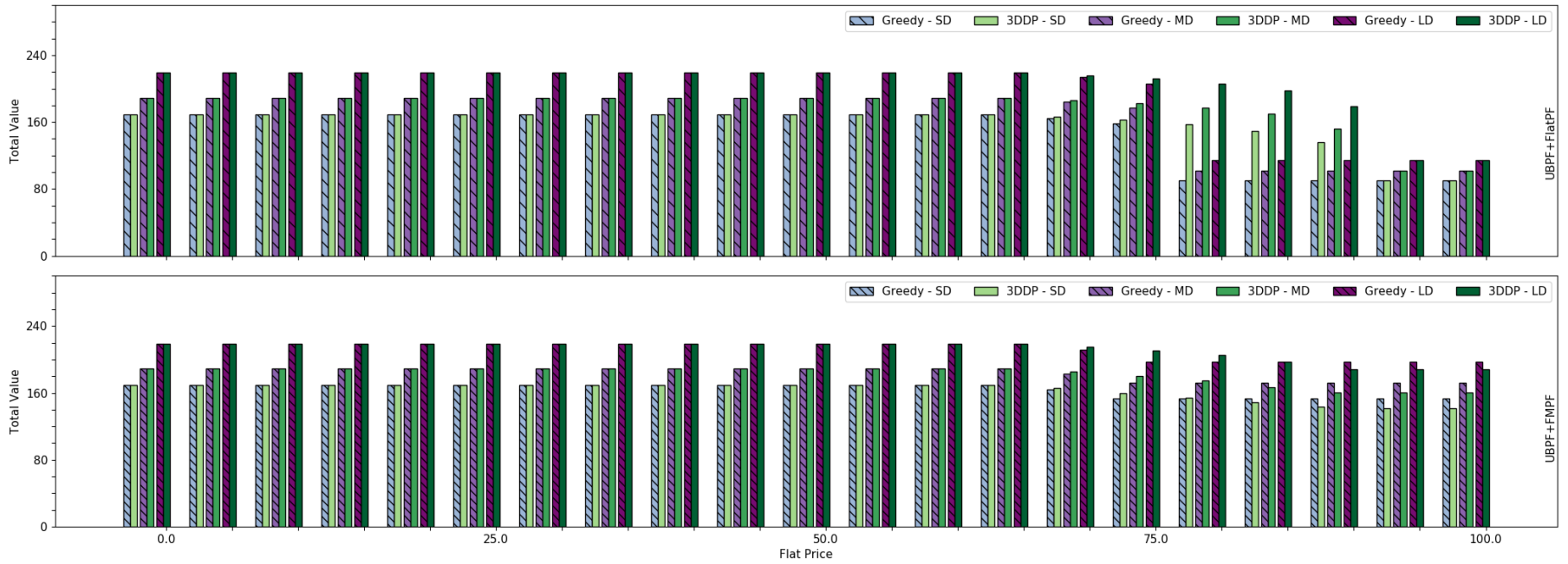


FIGURE A.9: Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD3 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities

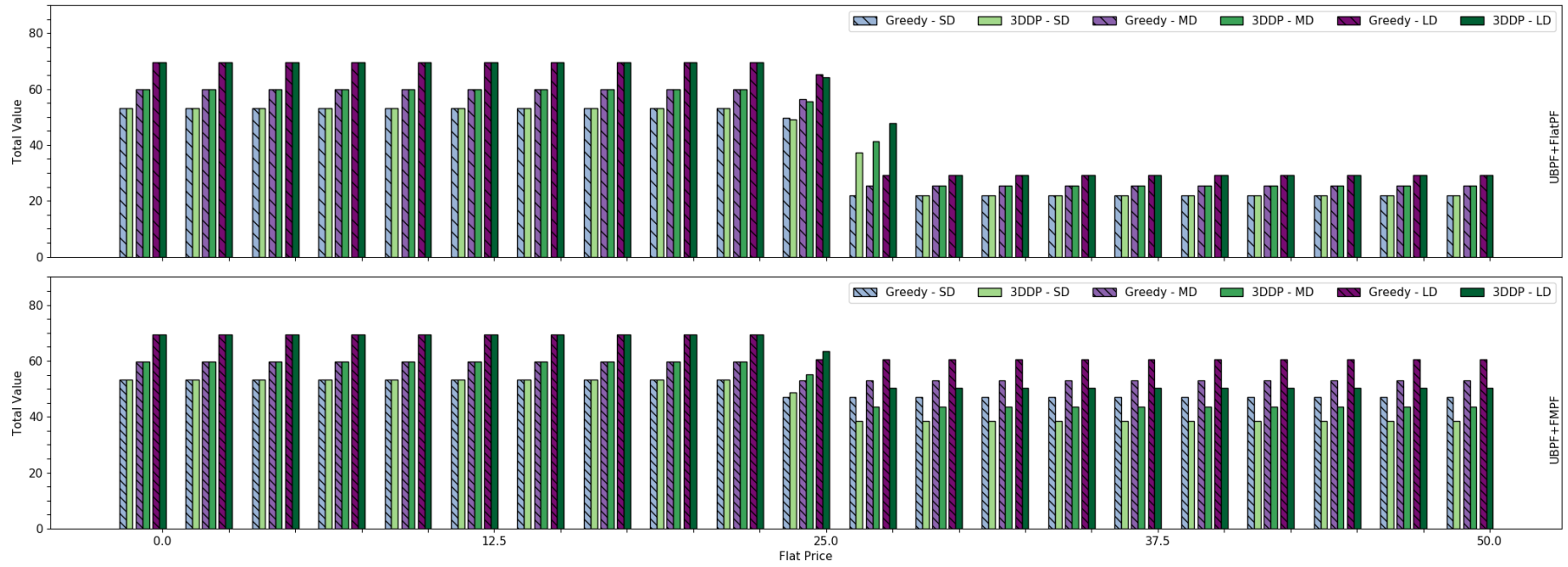


FIGURE A.10: Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD4 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities

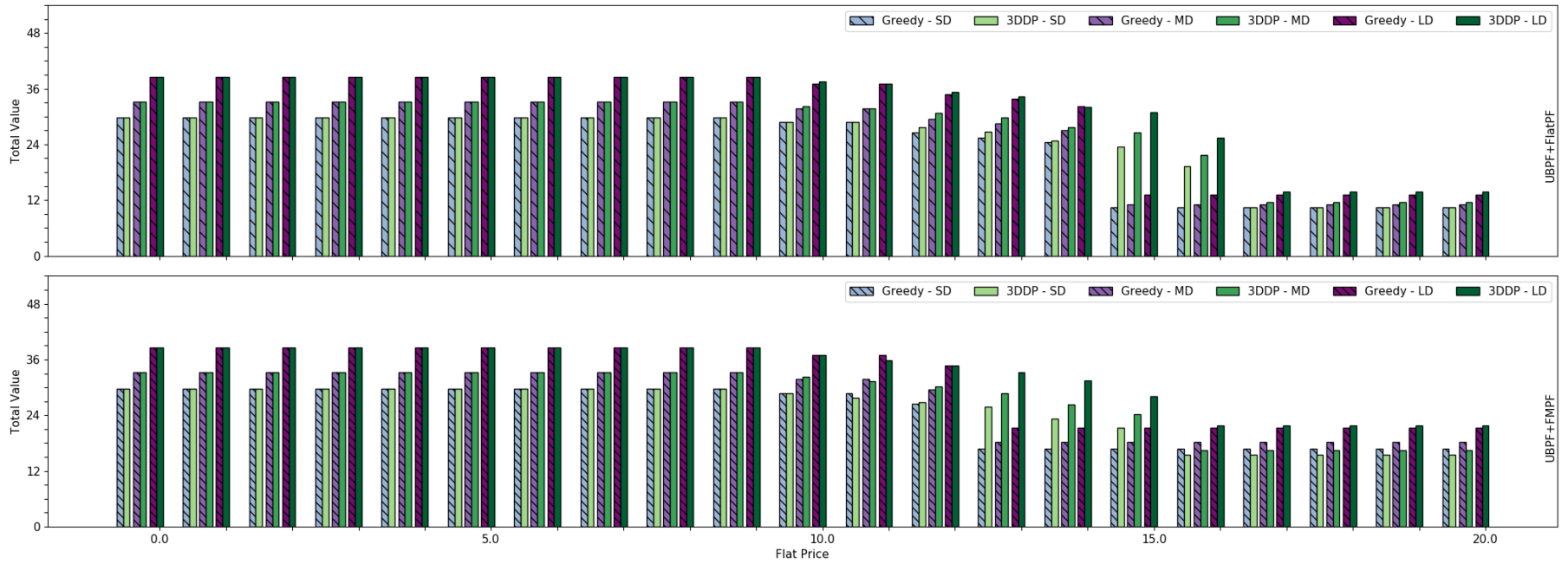


FIGURE A.11: Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD5 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities

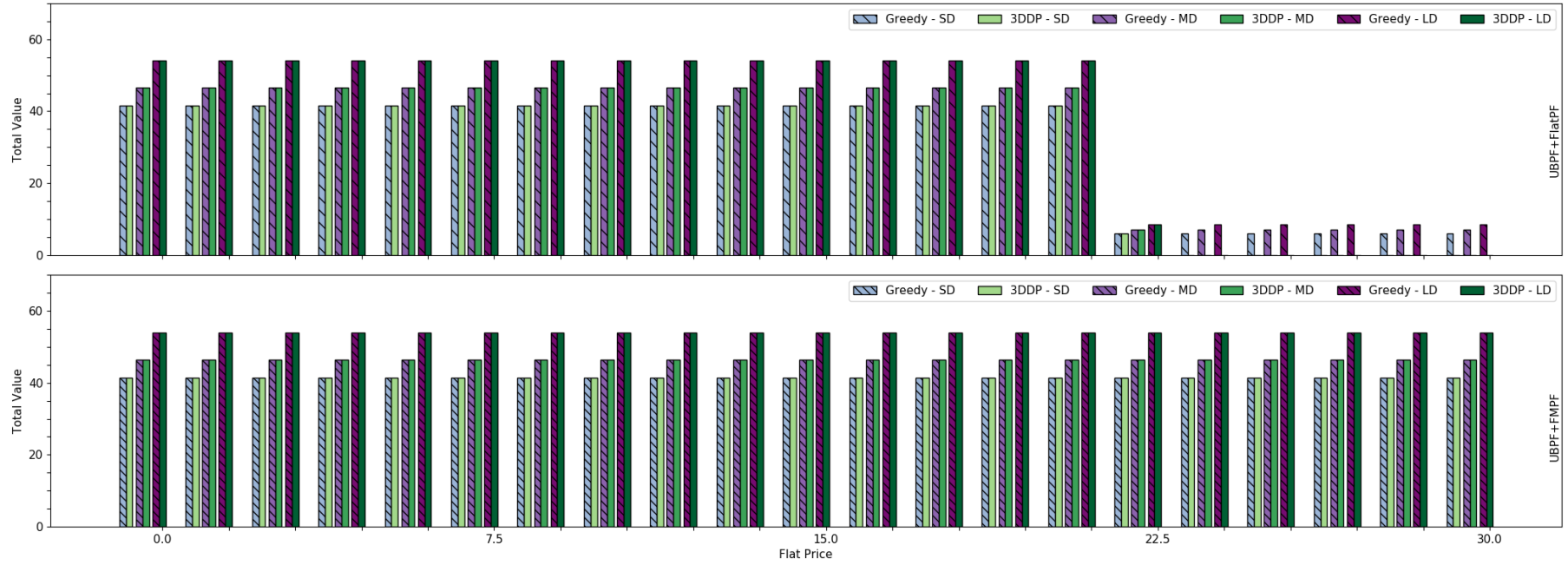


FIGURE A.12: Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD6 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities

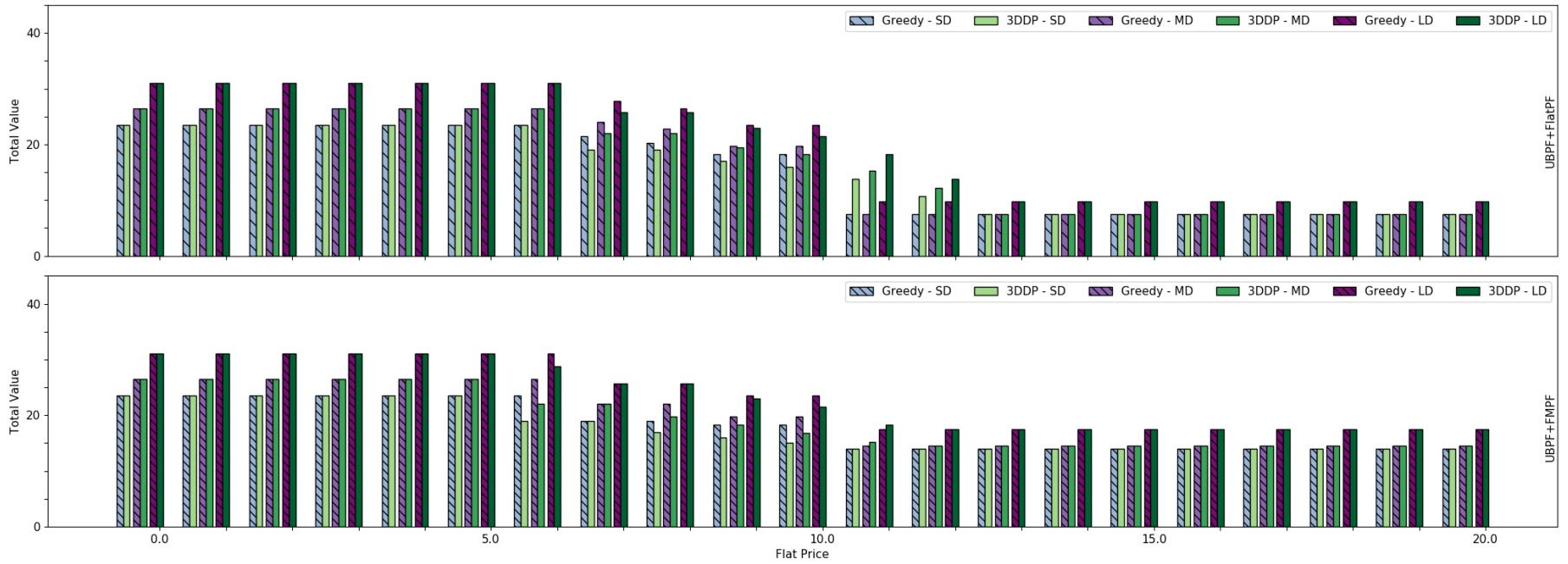


FIGURE A.13: Total Utility of the Allocations Returned by Greedy and 3DDP for Query LD8 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities

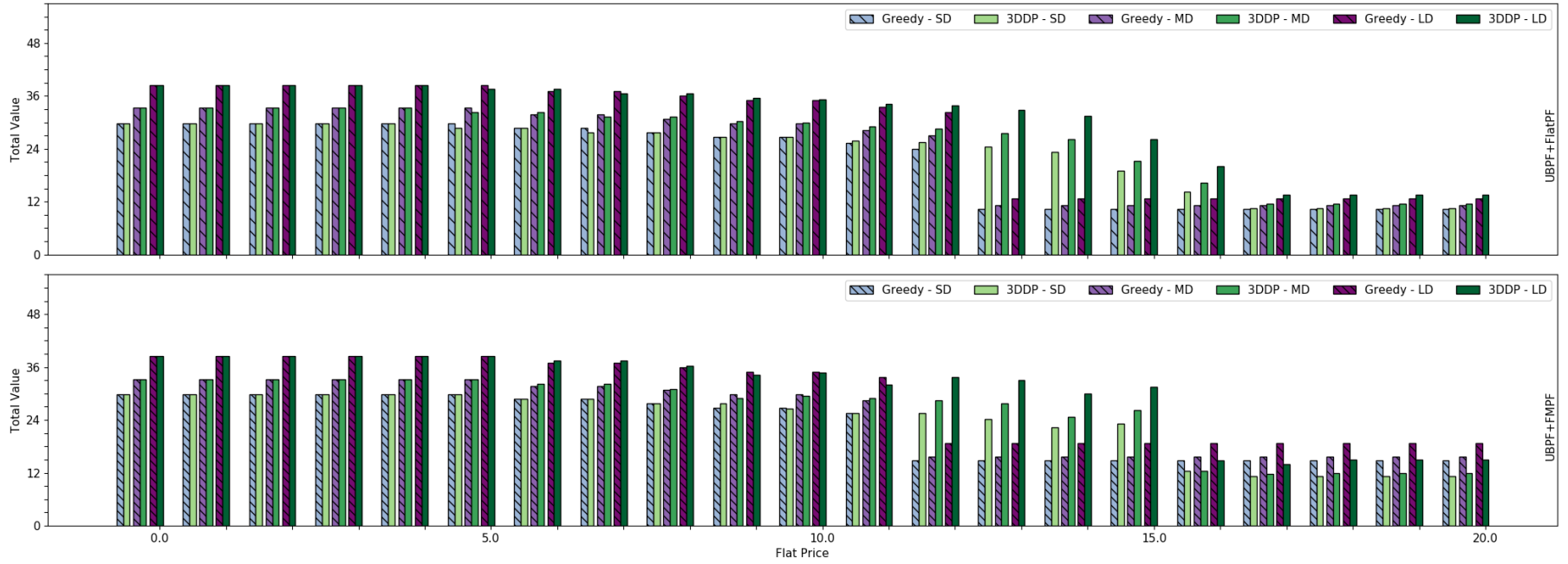


FIGURE A.14: Total Utility of the Allocations Returned by Greedy and 3DDP for Query LS6 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities

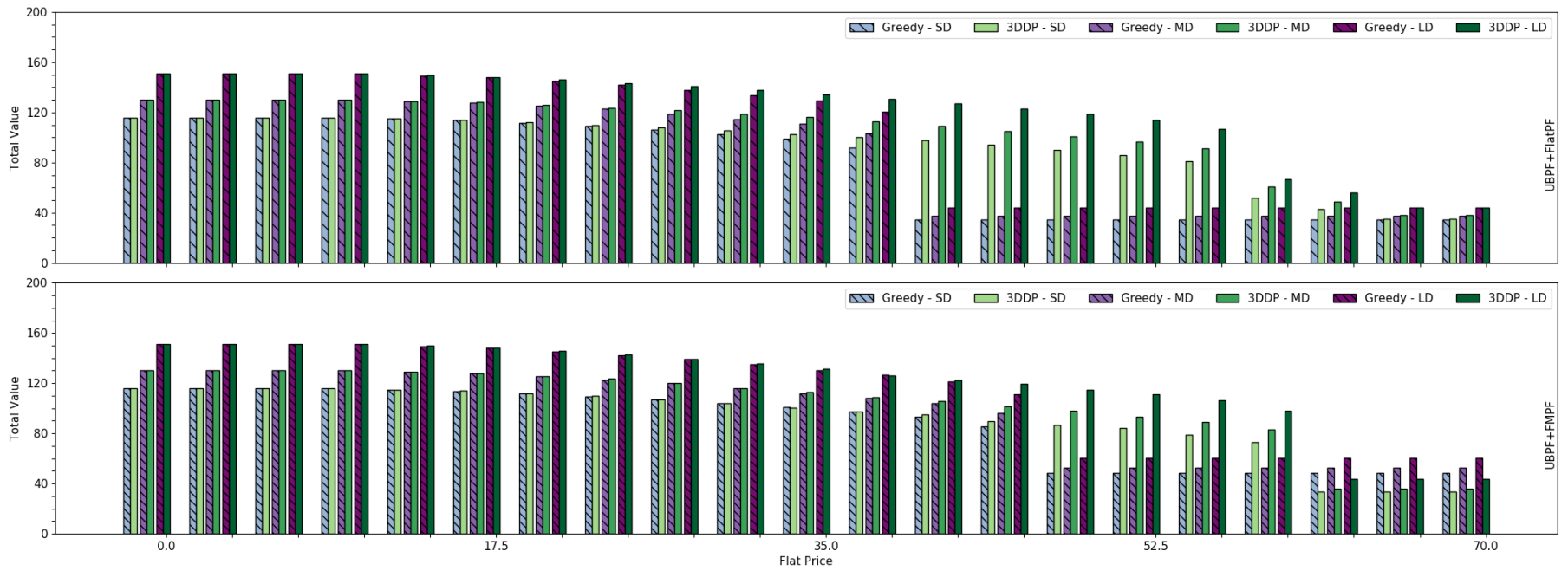


FIGURE A.15: Total Utility of the Allocations Returned by Greedy and 3DDP for Query LS7 over FlatPF[P] and FMPF[P, N] with Different P under Sized-Distributions of Utilities

Appendix B

Experiments Settings and Results of Evaluation in Chapter 6

B.1 Random Pricing Function Settings

TABLE B.1: 10 Random Settings of Freemium Pricing Function $f(X, L, p)$ for Data Sources

	ChEBI	DBpedia	DrugBank	KEGG	Geonames	Jamendo	LinkedMDB	NY Times	SW Dog Food
1	$f(42.94, 5802, 0.17)$	$f(9.35, 5024, 0.26)$	$f(74.75, 7460, 0.8)$	$f(0.87, 7959, 0.88)$	$f(3.43, 9532, 0.54)$	$f(93.03, 7579, 0.58)$	$f(43.29, 8626, 0.72)$	$f(87.79, 175, 0.97)$	$f(54.47, 5839, 0.36)$
2	$f(89.38, 5928, 0.45)$	$f(38.1, 9880, 0.19)$	$f(97.67, 7005, 0.65)$	$f(83.58, 20, 0.92)$	$f(72.07, 1729, 0.34)$	$f(46.5, 9485, 0.65)$	$f(23.5, 4216, 0.68)$	$f(84.54, 2164, 0.48)$	$f(68.02, 1427, 0.77)$
3	$f(62.01, 3763, 0.09)$	$f(18.13, 9799, 0.32)$	$f(91.03, 3848, 0.38)$	$f(53.71, 278, 0.12)$	$f(55.15, 413, 0.33)$	$f(48.19, 9756, 0.24)$	$f(32.76, 6025, 0.21)$	$f(11.41, 1845, 0.46)$	$f(46.59, 8916, 0.3)$
4	$f(74.56, 6398, 0.58)$	$f(12.54, 9270, 0.7)$	$f(36.6, 8056, 0.56)$	$f(69.33, 2284, 0.45)$	$f(78.95, 7405, 0.98)$	$f(37.1, 2046, 0.8)$	$f(58.39, 4635, 0.6)$	$f(82.17, 4185, 0.36)$	$f(24.85, 8890, 0.77)$
5	$f(33.37, 4922, 0.78)$	$f(3.76, 102, 0.99)$	$f(49.37, 4939, 0.68)$	$f(75.69, 6242, 0.75)$	$f(9.95, 5270, 0.22)$	$f(2.85, 5515, 0.22)$	$f(55.69, 2939, 0.27)$	$f(87.33, 967, 0.02)$	$f(41.42, 8681, 0.74)$
6	$f(67.62, 9077, 0.82)$	$f(77.86, 6268, 0.64)$	$f(28.44, 2572, 0.68)$	$f(25.8, 9113, 0.59)$	$f(42.85, 3806, 0.62)$	$f(36.29, 6022, 0.74)$	$f(6.51, 8594, 0.57)$	$f(81.22, 9889, 0.89)$	$f(29.37, 3300, 0.74)$
7	$f(92.13, 8712, 0.3)$	$f(28.39, 9659, 0.04)$	$f(24.92, 204, 0.86)$	$f(22.21, 736, 0.7)$	$f(77.73, 3082, 0.46)$	$f(11.39, 4574, 0.3)$	$f(23.64, 9435, 0.55)$	$f(65.33, 9161, 0.83)$	$f(38.65, 858, 0.62)$
8	$f(57.33, 1811, 0.27)$	$f(41.22, 9772, 0.6)$	$f(78.08, 4515, 0.02)$	$f(20.05, 8138, 0.6)$	$f(20.34, 1395, 0.03)$	$f(19.41, 4137, 0.12)$	$f(40.22, 6044, 0.83)$	$f(38.66, 955, 0.27)$	$f(36.02, 885, 0.32)$
9	$f(81.55, 4436, 0.98)$	$f(23.76, 5359, 0.07)$	$f(79.0, 1238, 0.86)$	$f(4.07, 6726, 0.99)$	$f(74.93, 6393, 0.43)$	$f(1.33, 4357, 0.67)$	$f(58.15, 4233, 0.46)$	$f(75.3, 8299, 0.95)$	$f(82.34, 3662, 0.52)$
10	$f(30.59, 6227, 0.28)$	$f(35.79, 5820, 0.89)$	$f(52.73, 9283, 0.18)$	$f(52.39, 8514, 0.32)$	$f(63.51, 2583, 0.01)$	$f(10.27, 3347, 0.68)$	$f(17.28, 7919, 0.96)$	$f(74.03, 3946, 0.26)$	$f(44.58, 1129, 0.52)$

TABLE B.2: 10 Random Settings of Prefixed Pricing Function $f(0, 0, p)$ for Data Sources

	ChEBI	DBpedia	DrugBank	KEGG	Geonames	Jamendo	LinkedMDB	NY Times	SW Dog Food
1	$f(0, 0, 0.04)$	$f(0, 0, 0.13)$	$f(0, 0, 0.92)$	$f(0, 0, 0.7)$	$f(0, 0, 0.64)$	$f(0, 0, 0.01)$	$f(0, 0, 0.81)$	$f(0, 0, 0.55)$	$f(0, 0, 0.47)$
2	$f(0, 0, 0.77)$	$f(0, 0, 0.85)$	$f(0, 0, 0.78)$	$f(0, 0, 0.65)$	$f(0, 0, 0.19)$	$f(0, 0, 0.25)$	$f(0, 0, 0.52)$	$f(0, 0, 0.29)$	$f(0, 0, 0.96)$
3	$f(0, 0, 0.49)$	$f(0, 0, 0.6)$	$f(0, 0, 0.6)$	$f(0, 0, 0.09)$	$f(0, 0, 0.71)$	$f(0, 0, 0.14)$	$f(0, 0, 0.64)$	$f(0, 0, 0.4)$	$f(0, 0, 0.41)$
4	$f(0, 0, 0.04)$	$f(0, 0, 0.53)$	$f(0, 0, 0.76)$	$f(0, 0, 0.88)$	$f(0, 0, 0.51)$	$f(0, 0, 0.04)$	$f(0, 0, 0.94)$	$f(0, 0, 0.7)$	$f(0, 0, 0.33)$
5	$f(0, 0, 0.84)$	$f(0, 0, 0.88)$	$f(0, 0, 0.07)$	$f(0, 0, 0.05)$	$f(0, 0, 0.1)$	$f(0, 0, 0.88)$	$f(0, 0, 0.44)$	$f(0, 0, 0.19)$	$f(0, 0, 0.04)$
6	$f(0, 0, 0.95)$	$f(0, 0, 0.98)$	$f(0, 0, 0.36)$	$f(0, 0, 0.36)$	$f(0, 0, 0.98)$	$f(0, 0, 0.4)$	$f(0, 0, 0.08)$	$f(0, 0, 0.59)$	$f(0, 0, 0.13)$
7	$f(0, 0, 0.91)$	$f(0, 0, 0.73)$	$f(0, 0, 0.54)$	$f(0, 0, 0.49)$	$f(0, 0, 0.67)$	$f(0, 0, 0.06)$	$f(0, 0, 0.38)$	$f(0, 0, 0.39)$	$f(0, 0, 0.4)$
8	$f(0, 0, 0.16)$	$f(0, 0, 0.91)$	$f(0, 0, 0.49)$	$f(0, 0, 0.67)$	$f(0, 0, 0.56)$	$f(0, 0, 0.42)$	$f(0, 0, 0.74)$	$f(0, 0, 0.54)$	$f(0, 0, 0.96)$
9	$f(0, 0, 0.72)$	$f(0, 0, 0.48)$	$f(0, 0, 0.66)$	$f(0, 0, 0.51)$	$f(0, 0, 0.1)$	$f(0, 0, 0.34)$	$f(0, 0, 0.74)$	$f(0, 0, 0.11)$	$f(0, 0, 0.81)$
10	$f(0, 0, 0.4)$	$f(0, 0, 0.86)$	$f(0, 0, 0.16)$	$f(0, 0, 0.45)$	$f(0, 0, 0.25)$	$f(0, 0, 0.34)$	$f(0, 0, 0.57)$	$f(0, 0, 0.17)$	$f(0, 0, 0.18)$

TABLE B.3: 10 Random Settings of Flat Pricing Function $f(X, \infty, 0)$ for Data Sources

	ChEBI	DBpedia	DrugBank	KEGG	Geonames	Jamendo	LinkedMDB	NY Times	SW Dog Food
1	$f(71.95, \infty, 0)$	$f(67.99, \infty, 0)$	$f(10.82, \infty, 0)$	$f(93.98, \infty, 0)$	$f(37.35, \infty, 0)$	$f(41.76, \infty, 0)$	$f(94.62, \infty, 0)$	$f(93.99, \infty, 0)$	$f(8.14, \infty, 0)$
2	$f(90.83, \infty, 0)$	$f(90.26, \infty, 0)$	$f(53.91, \infty, 0)$	$f(37.96, \infty, 0)$	$f(82.51, \infty, 0)$	$f(6.11, \infty, 0)$	$f(75.83, \infty, 0)$	$f(77.85, \infty, 0)$	$f(11.79, \infty, 0)$
3	$f(73.64, \infty, 0)$	$f(21.16, \infty, 0)$	$f(61.57, \infty, 0)$	$f(67.85, \infty, 0)$	$f(54.0, \infty, 0)$	$f(84.18, \infty, 0)$	$f(85.33, \infty, 0)$	$f(2.47, \infty, 0)$	$f(56.82, \infty, 0)$
4	$f(50.86, \infty, 0)$	$f(21.87, \infty, 0)$	$f(34.21, \infty, 0)$	$f(5.49, \infty, 0)$	$f(88.0, \infty, 0)$	$f(34.01, \infty, 0)$	$f(72.45, \infty, 0)$	$f(23.84, \infty, 0)$	$f(77.74, \infty, 0)$
5	$f(75.51, \infty, 0)$	$f(94.14, \infty, 0)$	$f(63.69, \infty, 0)$	$f(64.39, \infty, 0)$	$f(52.1, \infty, 0)$	$f(12.09, \infty, 0)$	$f(82.04, \infty, 0)$	$f(88.74, \infty, 0)$	$f(29.22, \infty, 0)$
6	$f(79.84, \infty, 0)$	$f(2.92, \infty, 0)$	$f(66.01, \infty, 0)$	$f(18.2, \infty, 0)$	$f(99.61, \infty, 0)$	$f(65.95, \infty, 0)$	$f(4.85, \infty, 0)$	$f(1.43, \infty, 0)$	$f(73.48, \infty, 0)$
7	$f(88.52, \infty, 0)$	$f(42.49, \infty, 0)$	$f(37.8, \infty, 0)$	$f(58.94, \infty, 0)$	$f(37.92, \infty, 0)$	$f(75.15, \infty, 0)$	$f(87.58, \infty, 0)$	$f(96.66, \infty, 0)$	$f(64.9, \infty, 0)$
8	$f(34.96, \infty, 0)$	$f(31.84, \infty, 0)$	$f(6.73, \infty, 0)$	$f(17.32, \infty, 0)$	$f(88.32, \infty, 0)$	$f(15.36, \infty, 0)$	$f(43.55, \infty, 0)$	$f(87.4, \infty, 0)$	$f(65.54, \infty, 0)$
9	$f(78.26, \infty, 0)$	$f(70.52, \infty, 0)$	$f(89.14, \infty, 0)$	$f(95.03, \infty, 0)$	$f(42.86, \infty, 0)$	$f(25.58, \infty, 0)$	$f(28.19, \infty, 0)$	$f(99.37, \infty, 0)$	$f(66.76, \infty, 0)$
10	$f(57.54, \infty, 0)$	$f(97.8, \infty, 0)$	$f(15.21, \infty, 0)$	$f(47.26, \infty, 0)$	$f(70.29, \infty, 0)$	$f(76.74, \infty, 0)$	$f(63.57, \infty, 0)$	$f(82.75, \infty, 0)$	$f(52.68, \infty, 0)$

B.2 Query Plans

The following tables list the stack cost of the query plans generated by different algorithms at the corresponding pricing function settings. Each every colourful bar marks the join-selectivity of a triple pattern over a data source and its cost in the order of execution.

TABLE B.4: Query Plans of Query CD1 at Settings of 10 Flat Pricing Functions

No.	Query Plans		
#1	CostFed		
	Gen-Greedy		
	Sum-Greedy		
#2	CostFed		
#3	CostFed		
	Gen-Greedy		
	Sum-Greedy		
#4	CostFed		
	Gen-Greedy		
			Continued on next page

Table B.4 – continued from previous page

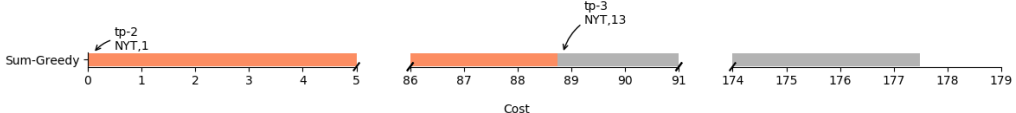
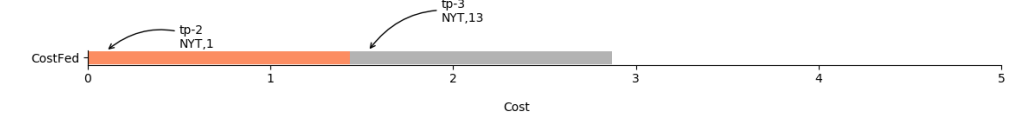
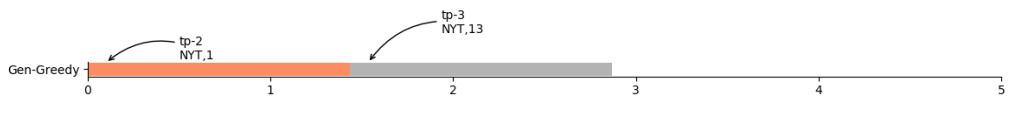
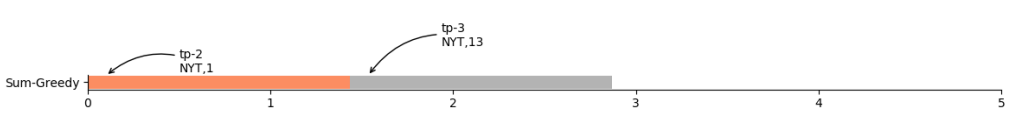
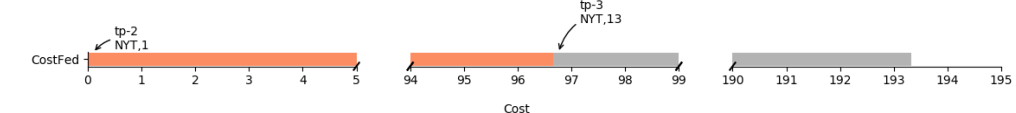
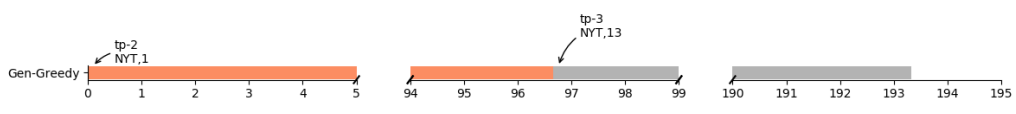
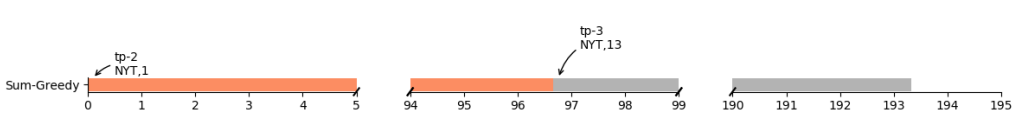
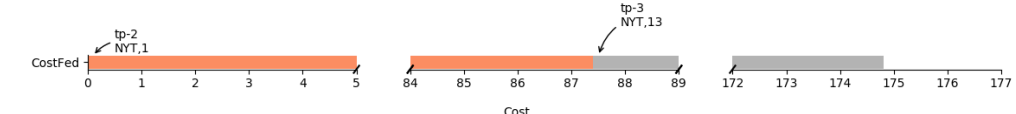
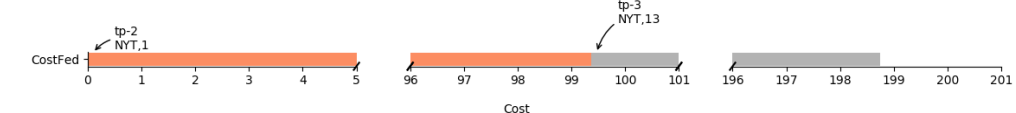
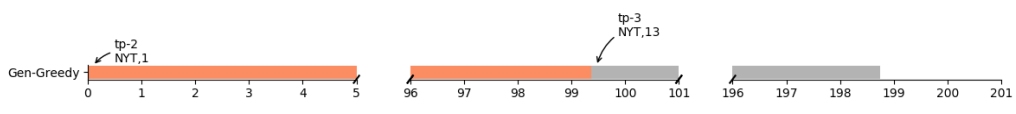
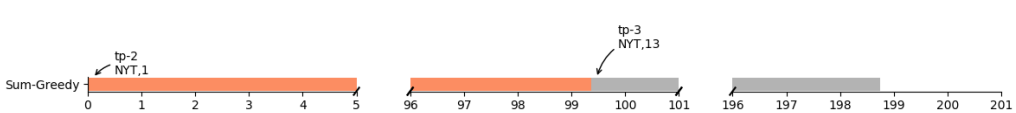
No.	Query Plans
	 <p>Sum-Greedy</p> <p>Cost</p>
#5	 <p>CostFed</p> <p>Cost</p>  <p>Gen-Greedy</p> <p>Cost</p>  <p>Sum-Greedy</p> <p>Cost</p>
#6	 <p>CostFed</p> <p>Cost</p>  <p>Gen-Greedy</p> <p>Cost</p>  <p>Sum-Greedy</p> <p>Cost</p>
#7	 <p>CostFed</p> <p>Cost</p>
#8	 <p>CostFed</p> <p>Cost</p>  <p>Gen-Greedy</p> <p>Cost</p>  <p>Sum-Greedy</p> <p>Cost</p>
Continued on next page	

Table B.4 – continued from previous page

No.	Query Plans
#9	<p>CostFed</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>tp-3 NYT,13</p> <p>Cost</p>
#10	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>tp-3 NYT,13</p> <p>Cost</p>

TABLE B.5: Query Plans of Query CD1 at Settings of 10 freemium Pricing Functions

No.	Query Plans
#1	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>tp-3 NYT,13</p> <p>Cost</p>
#2	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>tp-3 NYT,13</p> <p>Cost</p>

Continued on next page

Table B.5 – continued from previous page

No.	Query Plans
#3	<p>CostFed: tp-2 NYT,1 (0-5), tp-3 NYT,13 (79-84), tp-3 NYT,13 (161-166)</p> <p>Gen-Greedy: tp-2 NYT,1 (0-5), tp-3 NYT,13 (79-84), tp-3 NYT,13 (161-166)</p> <p>Sum-Greedy: tp-2 NYT,1 (0-5), tp-3 NYT,13 (79-84), tp-3 NYT,13 (161-166)</p>
#4	<p>CostFed: tp-2 NYT,1 (0-5), tp-3 NYT,13 (84-89), tp-3 NYT,13 (172-177)</p> <p>Gen-Greedy: tp-2 NYT,1 (0-5), tp-3 NYT,13 (84-89), tp-3 NYT,13 (172-177)</p> <p>Sum-Greedy: tp-2 NYT,1 (0-5), tp-3 NYT,13 (84-89), tp-3 NYT,13 (172-177)</p>
#5	<p>CostFed: tp-2 NYT,1 (0-5), tp-3 NYT,13 (78-83), tp-3 NYT,13 (159-164)</p> <p>Gen-Greedy: tp-2 NYT,1 (0-5), tp-3 NYT,13 (78-83), tp-3 NYT,13 (159-164)</p> <p>Sum-Greedy: tp-2 NYT,1 (0-5), tp-3 NYT,13 (78-83), tp-3 NYT,13 (159-164)</p>
#6	<p>CostFed: tp-2 NYT,1 (0-5), tp-3 NYT,13 (62-67), tp-3 NYT,13 (128-133)</p> <p>Gen-Greedy: tp-2 NYT,1 (0-5), tp-3 NYT,13 (62-67), tp-3 NYT,13 (128-133)</p>

Continued on next page

Table B.5 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>Cost</p>
#7	<p>CostFed</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>Cost</p> <p>Gen-Greedy</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>Cost</p> <p>Sum-Greedy</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>Cost</p>
#8	<p>CostFed</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>Cost</p> <p>Gen-Greedy</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>Cost</p> <p>Sum-Greedy</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>Cost</p>
#9	<p>CostFed</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>Cost</p> <p>Gen-Greedy</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>Cost</p> <p>Sum-Greedy</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>Cost</p>
#10	<p>CostFed</p> <p>tp-2 NYT,1</p> <p>tp-3 NYT,13</p> <p>Cost</p>
#10	Continued on next page

Table B.5 – continued from previous page

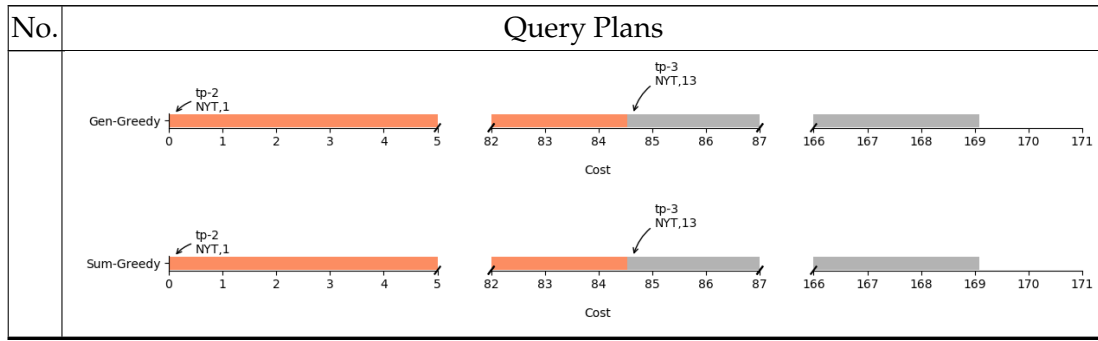
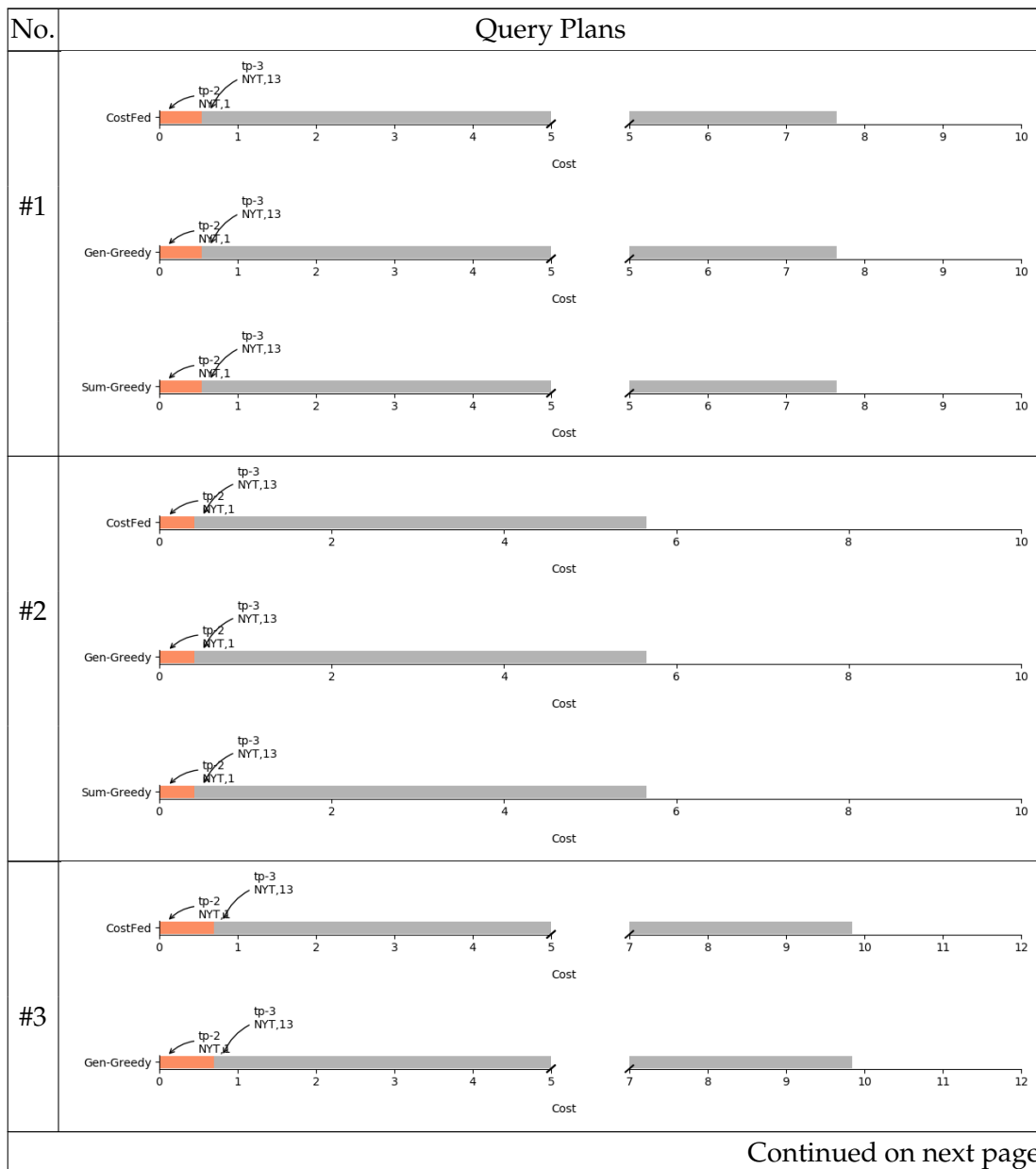


TABLE B.6: Query Plans of Query CD1 at Settings of 10 per Pricing Functions



Continued on next page

Table B.6 – continued from previous page

No.	Query Plans
#4	
#5	
#6	
#7	

Continued on next page

Table B.6 – continued from previous page

No.	Query Plans
	<p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#8	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#9	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#10	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>

TABLE B.7: Query Plans of Query CD2 at Settings of 10 Flat Pricing Functions

No.	Query Plans
#1	<p>CostFed: tp-2 NYT,7432 (0-5), tp-3 NYT,1 (91-96), tp-1 DBpedia,1 (185-190), 253-256, 257-258</p> <p>Gen-Greedy: tp-1 DBpedia,1 (0-5), tp-2 NYT,7432 (65-70), tp-3 NYT,1 (159-164), 253-256, 257-258</p> <p>Sum-Greedy: tp-1 DBpedia,1 (0-5), tp-2 NYT,7432 (65-70), tp-3 NYT,1 (159-164), 253-256, 257-258</p>
#2	<p>CostFed: tp-2 NYT,7432 (0-5), tp-3 NYT,1 (23-26), 27-28</p> <p>Gen-Greedy: tp-2 NYT,7432 (0-5), tp-3 NYT,1 (23-26), 27-28</p> <p>Sum-Greedy: tp-2 NYT,7432 (0-5), tp-3 NYT,1 (23-26), 27-28</p>
#3	<p>CostFed: tp-2 NYT,7432 (0-5), tp-3 NYT,1 (21-26), tp-1 DBpedia,1 (45-50), 67-72</p> <p>Gen-Greedy: tp-1 DBpedia,1 (0-5), tp-2 NYT,7432 (19-24), tp-3 NYT,1 (43-48), 67-72</p> <p>Sum-Greedy: tp-1 DBpedia,1 (0-5), tp-2 NYT,7432 (19-24), tp-3 NYT,1 (43-48), 67-72</p>
#4	<p>CostFed: tp-2 NYT,7432 (0-5), tp-3 NYT,1 (86-91), tp-1 DBpedia,1 (174-179), 269-274</p> <p>Gen-Greedy: tp-2 NYT,7432 (0-5), tp-3 NYT,1 (86-91), tp-1 DBpedia,1 (174-179), 269-274</p>

Continued on next page

Table B.7 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#5	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#6	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#8	<p>CostFed</p> <p>Cost</p>

Continued on next page

Table B.7 – continued from previous page

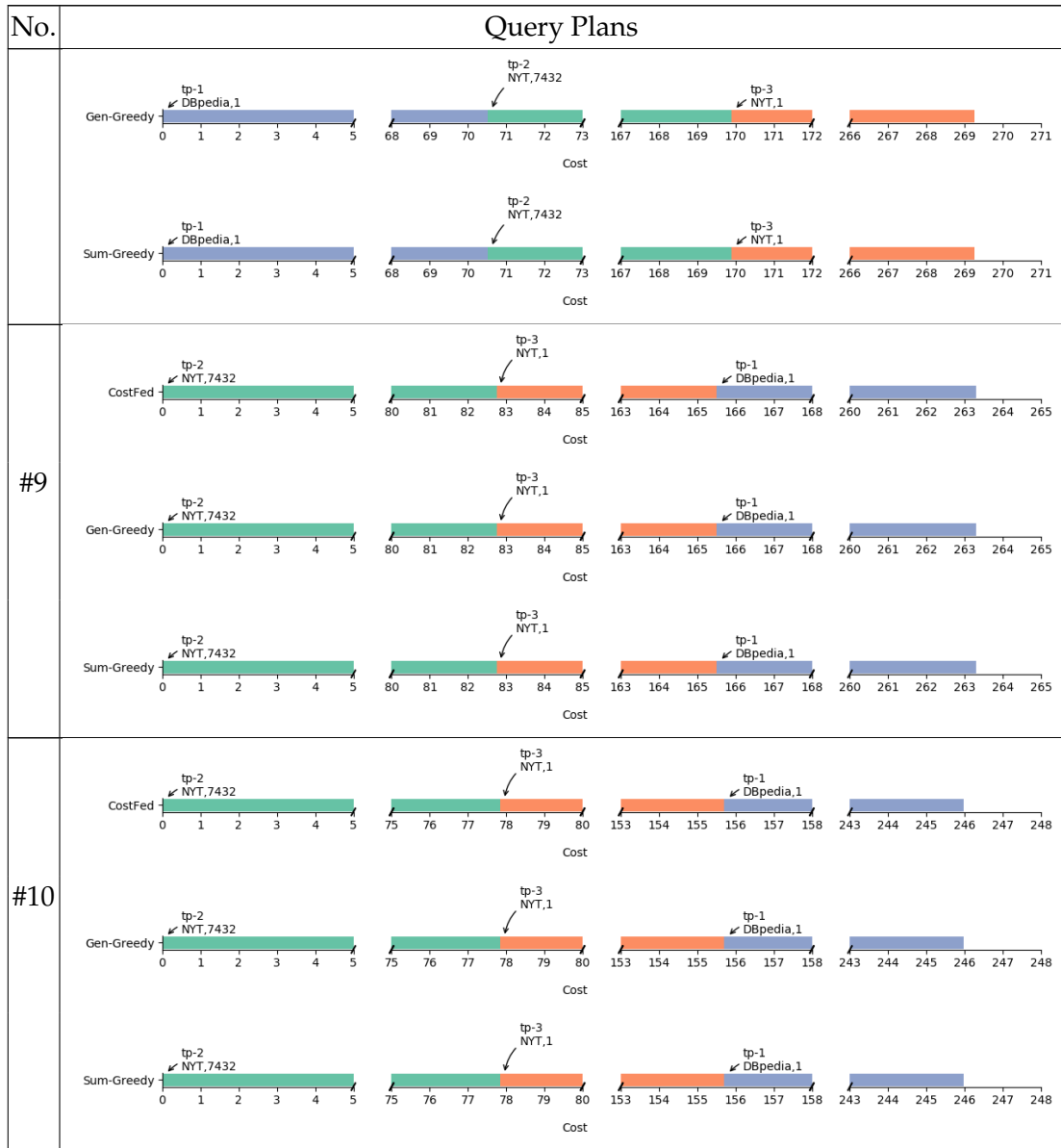
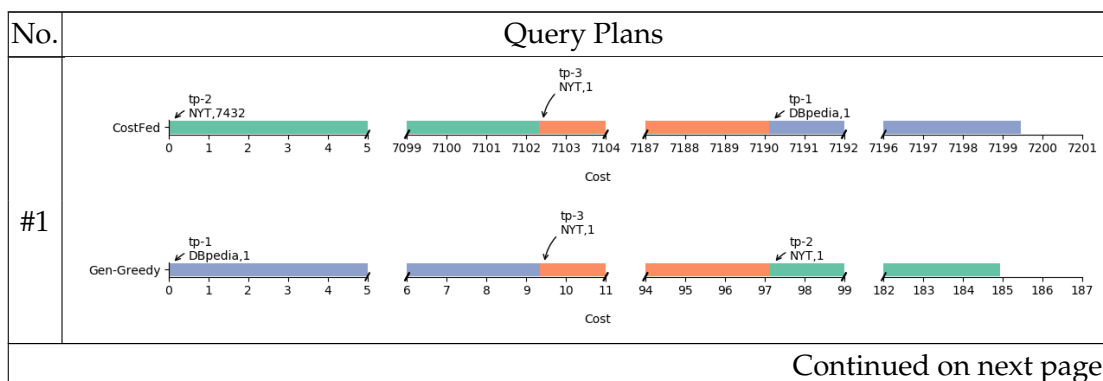


TABLE B.8: Query Plans of Query CD2 at Settings of 10 freemium Pricing Functions



Continued on next page

Table B.8 – continued from previous page

No.	Query Plans
#2	
#3	
#4	
#5	Continued on next page

Table B.8 – continued from previous page

No.	Query Plans
	<p>Gen-Greedy</p> <p>Cost</p>
#6	<p>Sum-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Cost</p>
#8	<p>Gen-Greedy</p> <p>Cost</p>
	<p>Sum-Greedy</p> <p>Cost</p>

Continued on next page

Table B.8 – continued from previous page

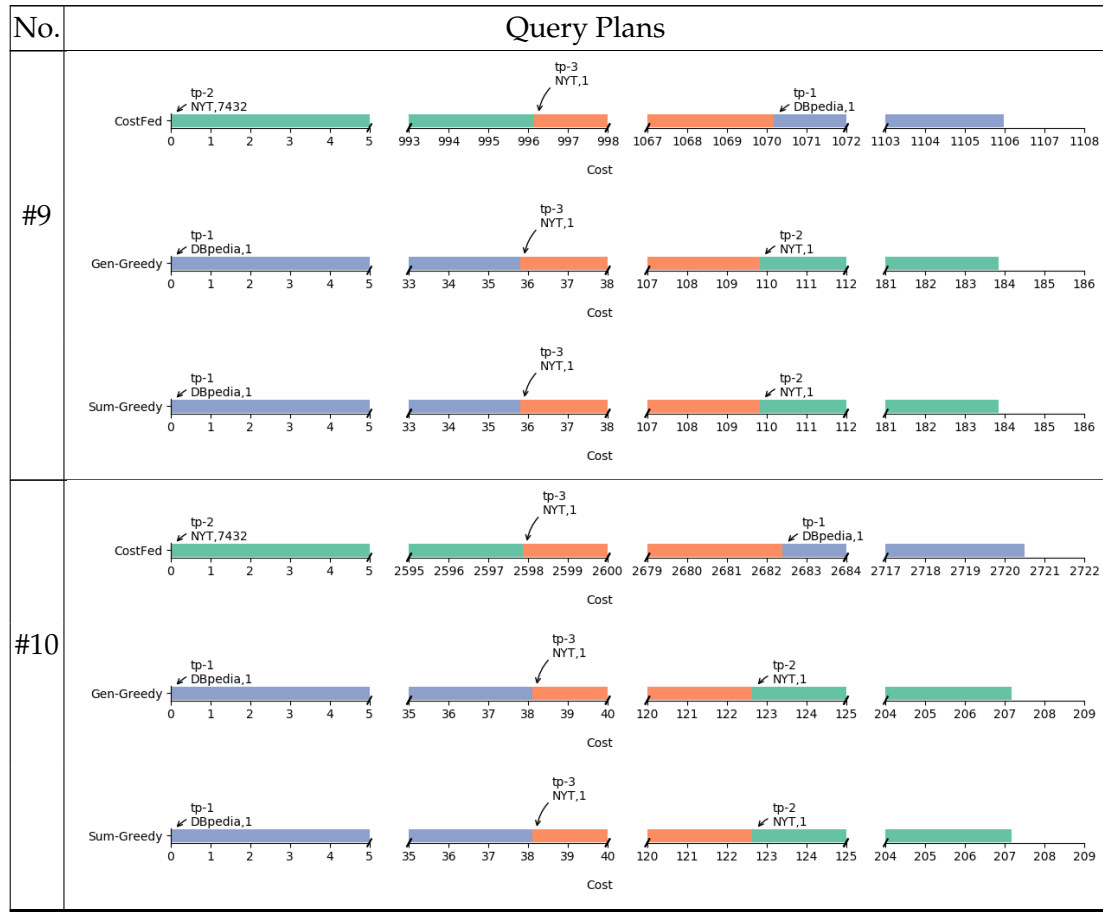
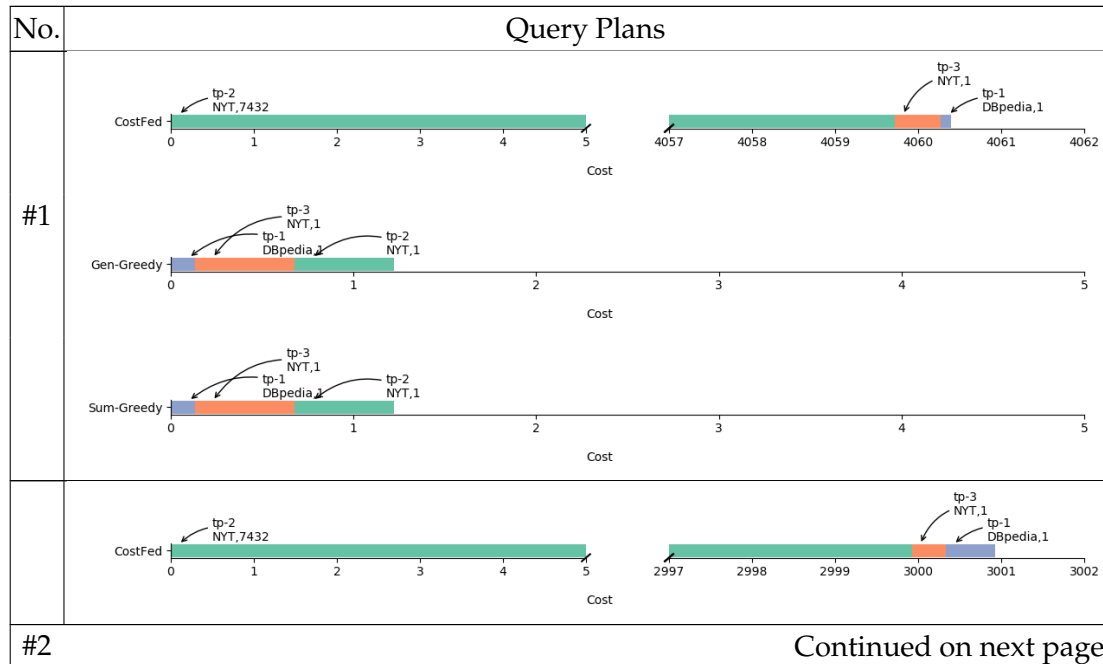


TABLE B.9: Query Plans of Query CD2 at Settings of 10 per Pricing Functions



Continued on next page

Table B.9 – continued from previous page

No.	Query Plans
#3	
#4	
#5	
Continued on next page	

Table B.9 – continued from previous page

No.	Query Plans
#6	<p>Query plan for #6. The CostFed plan shows a single segment for tp-2 NYT,7432. The Gen-Greedy plan shows segments for tp-3 NYT,1, tp-1 DBpedia,1, and tp-2 NYT,1. The Sum-Greedy plan shows segments for tp-3 NYT,1, tp-2 NYT,1, and tp-1 DBpedia,1. The x-axis represents Cost, ranging from 0 to 5.</p>
#7	<p>Query plan for #7. The CostFed plan shows a single segment for tp-2 NYT,7432. The Gen-Greedy plan shows segments for tp-3 NYT,1, tp-1 DBpedia,1, and tp-2 NYT,1. The Sum-Greedy plan shows segments for tp-3 NYT,1, tp-2 NYT,1, and tp-1 DBpedia,1. The x-axis represents Cost, ranging from 0 to 5.</p>
#8	<p>Query plan for #8. The CostFed plan shows a single segment for tp-2 NYT,7432. The Gen-Greedy plan shows segments for tp-3 NYT,1, tp-1 DBpedia,1, and tp-2 NYT,1. The Sum-Greedy plan shows segments for tp-3 NYT,1, tp-2 NYT,1, and tp-1 DBpedia,1. The x-axis represents Cost, ranging from 0 to 5.</p>
#9	<p>Query plan for #9. The CostFed plan shows a single segment for tp-2 NYT,7432. The Gen-Greedy plan shows segments for tp-3 NYT,1, tp-1 DBpedia,1, and tp-2 NYT,1. The Sum-Greedy plan shows segments for tp-3 NYT,1, tp-2 NYT,1, and tp-1 DBpedia,1. The x-axis represents Cost, ranging from 0 to 5.</p>

Continued on next page

Table B.9 – continued from previous page

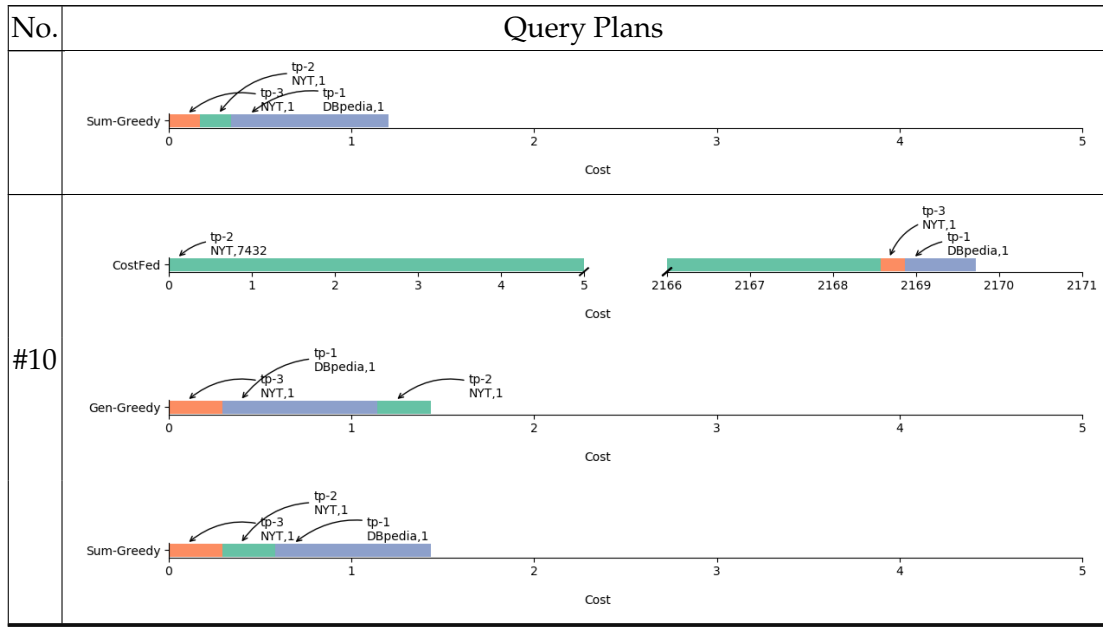
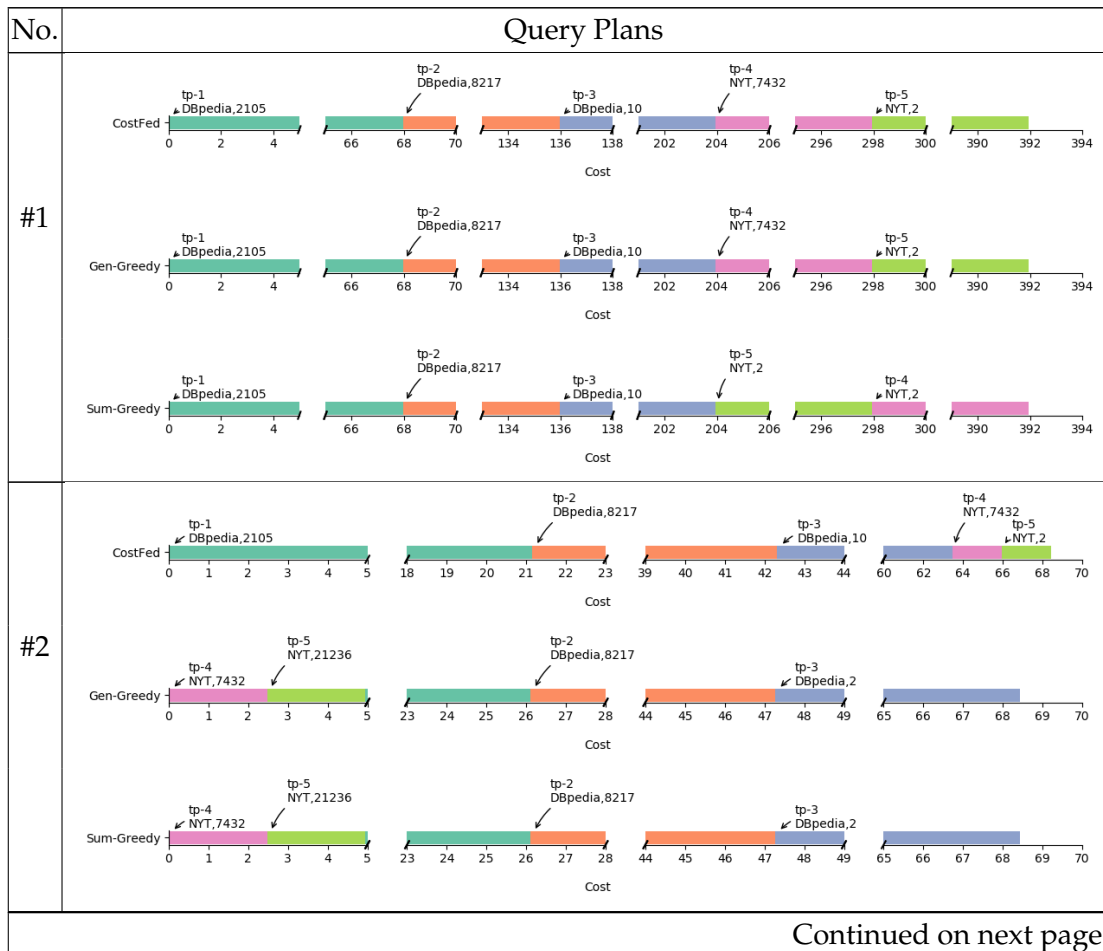


TABLE B.10: Query Plans of Query CD3 at Settings of 10 Flat Pricing Functions



Continued on next page

Table B.10 – continued from previous page

No.	Query Plans
#3	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#4	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#5	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#6	<p>CostFed</p> <p>Gen-Greedy</p> <p>Cost</p>

Continued on next page

Table B.10 – continued from previous page

No.	Query Plans
#7	
#8	
#9	
#10	

Continued on next page

Table B.10 – continued from previous page

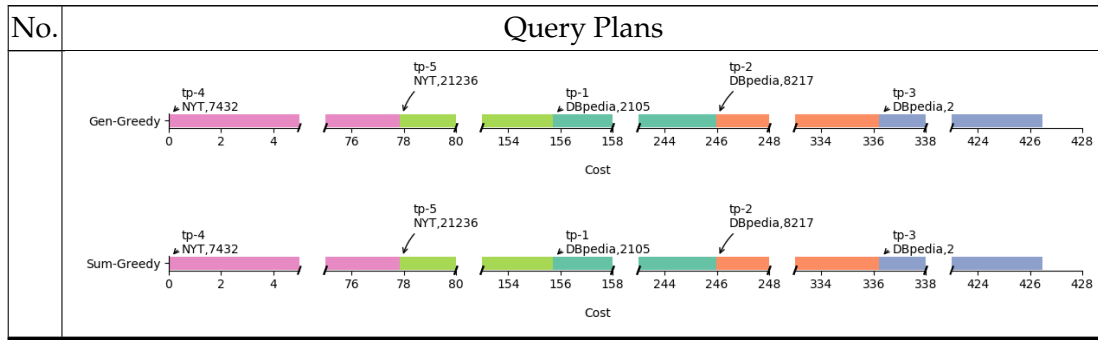
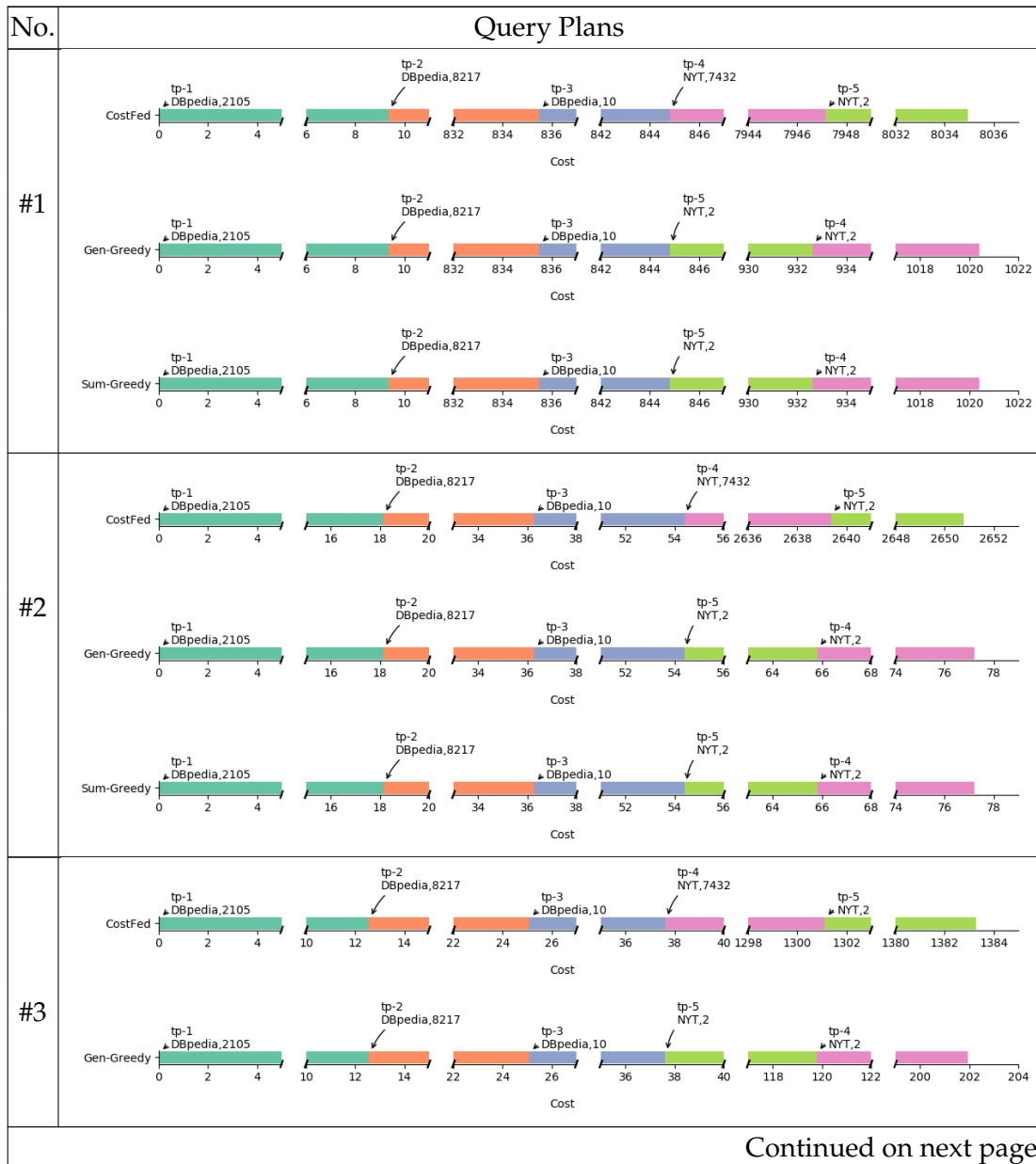


TABLE B.11: Query Plans of Query CD3 at Settings of 10 freemium Pricing Functions



Continued on next page

Table B.11 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#4	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#5	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#6	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Cost</p>

Continued on next page

Table B.11 – continued from previous page

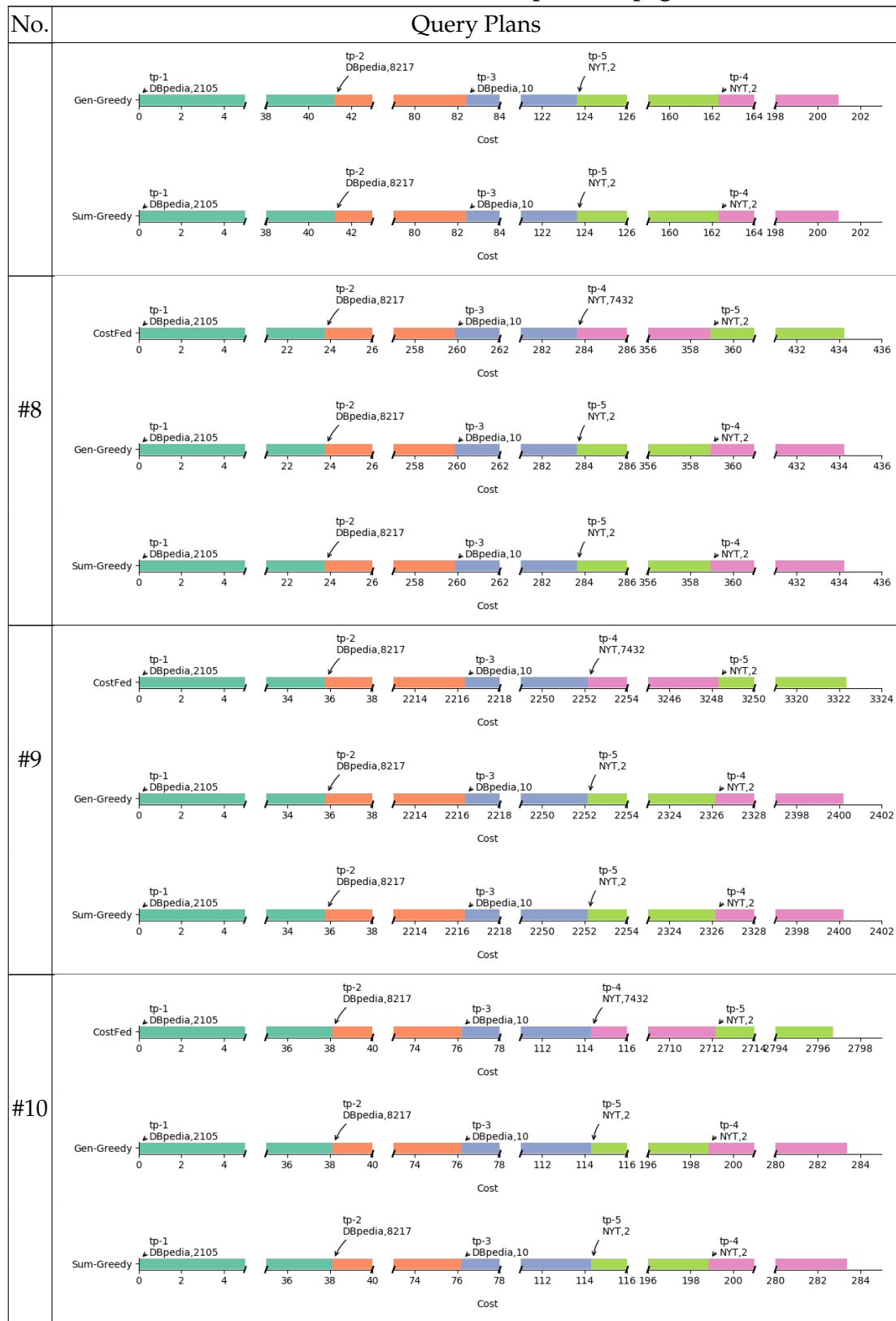


TABLE B.12: Query Plans of Query CD3 at Settings of 10 per Pricing Functions

No.	Query Plans
#1	<p>CostFed: 0 1 2 3 4 5 273 274 275 276 277 278 1349 1350 1351 1352 1353 1354 5410 5411 5412 5413 5414 5415</p> <p>Gen-Greedy: 0 1 2 3 4 5 273 274 275 276 277 278 1350 1352 1354 1356 1358</p> <p>Sum-Greedy: 0 1 2 3 4 5 273 274 275 276 277 278 1350 1352 1354 1356 1358</p>
#2	<p>CostFed: 0 2 4 1256 1258 1260 6174 6176 6178 6180 6182 6184 9180 9182 9184</p> <p>Gen-Greedy: 0 1 2 3 4 5 1256 1257 1258 1259 1260 1261 6173 6174 6175 6176 6177 6178 6179 6180 6181 6182 6183 6184</p> <p>Sum-Greedy: 0 1 2 3 4 5 1256 1257 1258 1259 1260 1261 6173 6174 6175 6176 6177 6178 6179 6180 6181 6182 6183 6184</p>
#3	<p>CostFed: 0 2 4 1104 1106 1108 5426 5428 5430 5430 5432 5434 10656 10658 10660</p> <p>Gen-Greedy: 0 1 2 3 4 5 1104 1105 1106 1107 1108 1109 5425 5426 5427 5428 5429 5430 5430 5432 5434 5436 5438 5440</p> <p>Sum-Greedy: 0 1 2 3 4 5 1104 1105 1106 1107 1108 1109 5425 5426 5427 5428 5429 5430 5430 5432 5434 5436 5438 5440</p>
#4	<p>CostFed: 0 2 4 1842 1844 1846 9038 9040 9042 9046 9048 9050 4 6 +1.049e4</p> <p>Gen-Greedy: 0 1 2 3 4 5 1841 1842 1843 1844 1845 1846 9038 9040 9042 9044 9046 9048</p>

Continued on next page

Table B.12 – continued from previous page

No.	Query Plans
#5	<p>CostFed: tp-1 DBpedia,2105 (0-4), tp-2 DBpedia,8217 (2064-2068), tp-3 DBpedia,10 (2-6), tp-4 NYT,7432 (2-6), tp-5 NYT,2 (0-4). Total Cost: +1.456e4.</p> <p>Gen-Greedy: tp-1 DBpedia,2105 (0-5), tp-2 DBpedia,8217 (2064-2069), tp-3 DBpedia,10 (1-6), tp-4 NYT,2 (10142-10146), tp-5 NYT,2 (10142-10150). Total Cost: +1.013e4.</p> <p>Sum-Greedy: tp-1 DBpedia,2105 (0-5), tp-2 DBpedia,8217 (2064-2069), tp-3 DBpedia,10 (1-6), tp-4 NYT,2 (10142-10146), tp-5 NYT,2 (10142-10150). Total Cost: +1.013e4.</p>
#6	<p>CostFed: tp-1 DBpedia,2105 (0-4), tp-2 DBpedia,8217 (1530-1534), tp-3 DBpedia,10 (7514-7518), tp-4 NYT,7432 (7522-7526), tp-5 NYT,2 (10416-10420). Total Cost: 10420.</p> <p>Gen-Greedy: tp-1 DBpedia,2105 (0-5), tp-2 DBpedia,8217 (1530-1535), tp-3 DBpedia,10 (7514-7519), tp-4 NYT,2 (7522-7526), tp-5 NYT,2 (7522-7527). Total Cost: 7527.</p> <p>Sum-Greedy: tp-1 DBpedia,2105 (0-5), tp-2 DBpedia,8217 (1530-1535), tp-3 DBpedia,10 (7514-7519), tp-4 NYT,2 (7522-7526), tp-5 NYT,2 (7522-7527). Total Cost: 7527.</p>
#7	<p>CostFed: tp-1 DBpedia,2105 (0-4), tp-2 DBpedia,8217 (1916-1920), tp-3 DBpedia,10 (9402-9406), tp-4 NYT,7432 (9412-9414), tp-5 NYT,2 (13416-13420). Total Cost: 13420.</p> <p>Gen-Greedy: tp-1 DBpedia,2105 (0-5), tp-2 DBpedia,8217 (1915-1920), tp-3 DBpedia,10 (9401-9406), tp-4 NYT,2 (9410-9414), tp-5 NYT,2 (9410-9415). Total Cost: 9415.</p> <p>Sum-Greedy: tp-1 DBpedia,2105 (0-5), tp-2 DBpedia,8217 (1915-1920), tp-3 DBpedia,10 (9401-9406), tp-4 NYT,2 (9410-9414), tp-5 NYT,2 (9410-9415). Total Cost: 9415.</p>
#8	<p>CostFed: tp-1 DBpedia,2105 (0-4), tp-2 DBpedia,8217 (1018-1022), tp-3 DBpedia,10 (5002-5006), tp-4 NYT,7432 (5008-5012), tp-5 NYT,2 (5792-5796). Total Cost: 5796.</p> <p>Gen-Greedy: tp-1 DBpedia,2105 (0-5), tp-2 DBpedia,8217 (1018-1023), tp-3 DBpedia,2 (5002-5006), tp-4 NYT,2 (5005-5007), tp-5 NYT,2 (5005-5007). Total Cost: 5007.</p>

Continued on next page

Table B.12 – continued from previous page

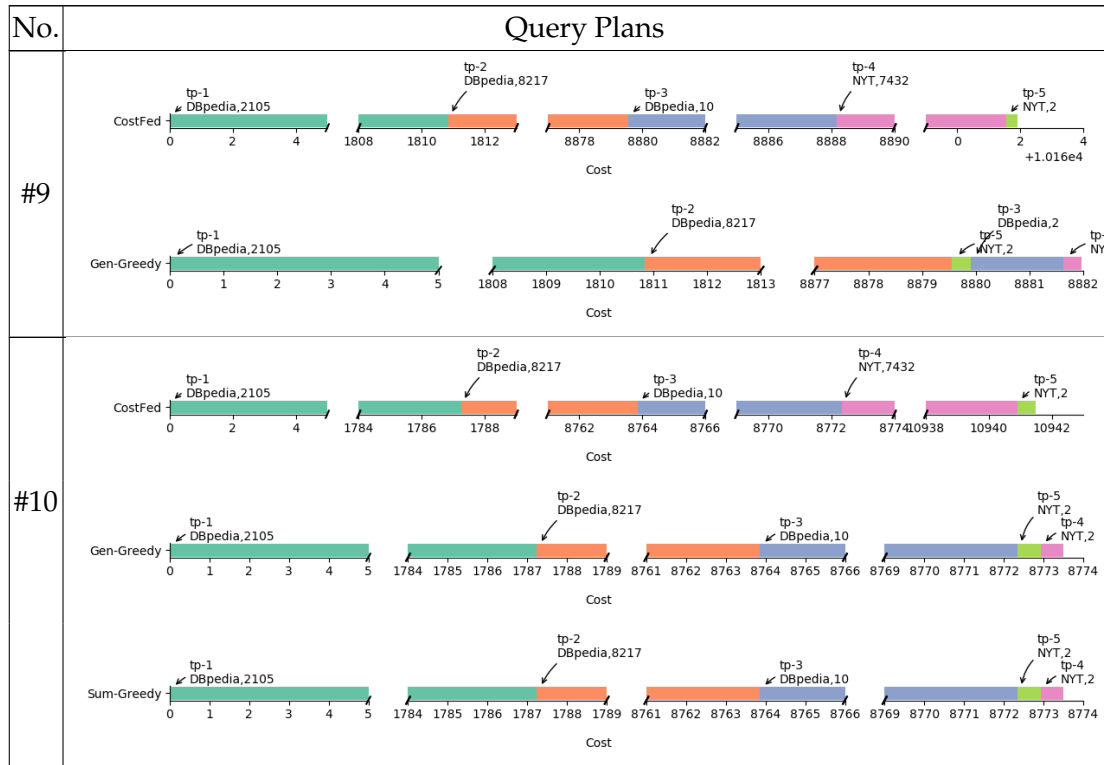
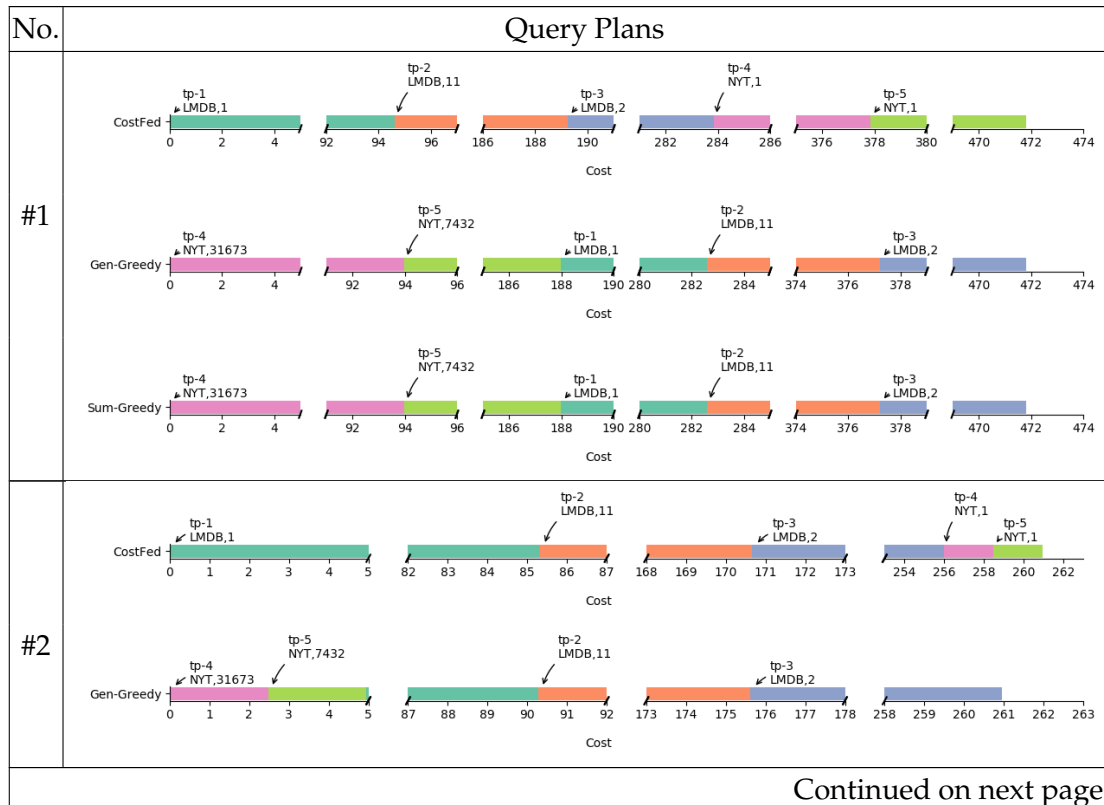


TABLE B.13: Query Plans of Query CD4 at Settings of 10 Flat Pricing Functions



Continued on next page

Table B.13 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#3	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#4	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#5	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#6	<p>CostFed</p> <p>Cost</p>

Continued on next page

Table B.13 – continued from previous page

No.	Query Plans
	<p>Gen-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#8	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#9	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
Continued on next page	

Table B.13 – continued from previous page

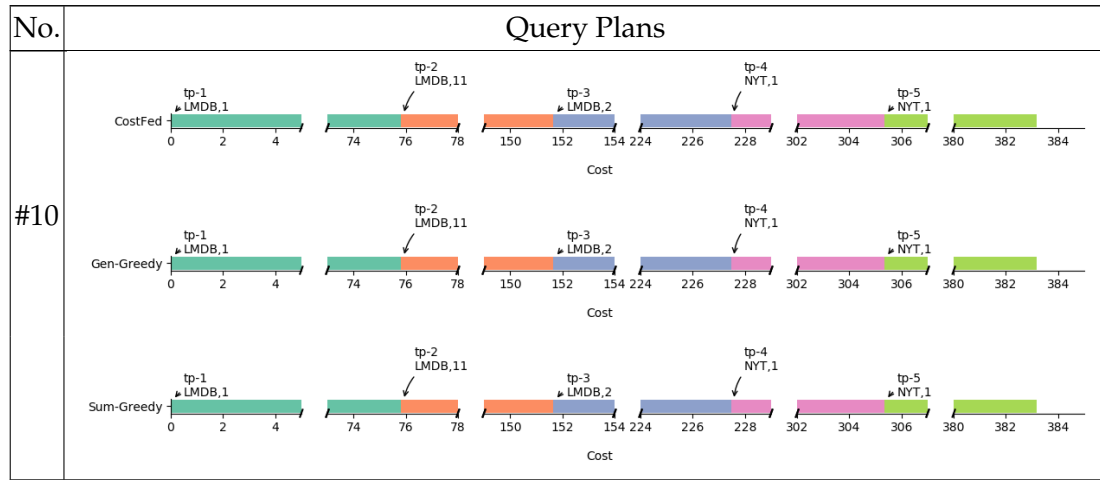
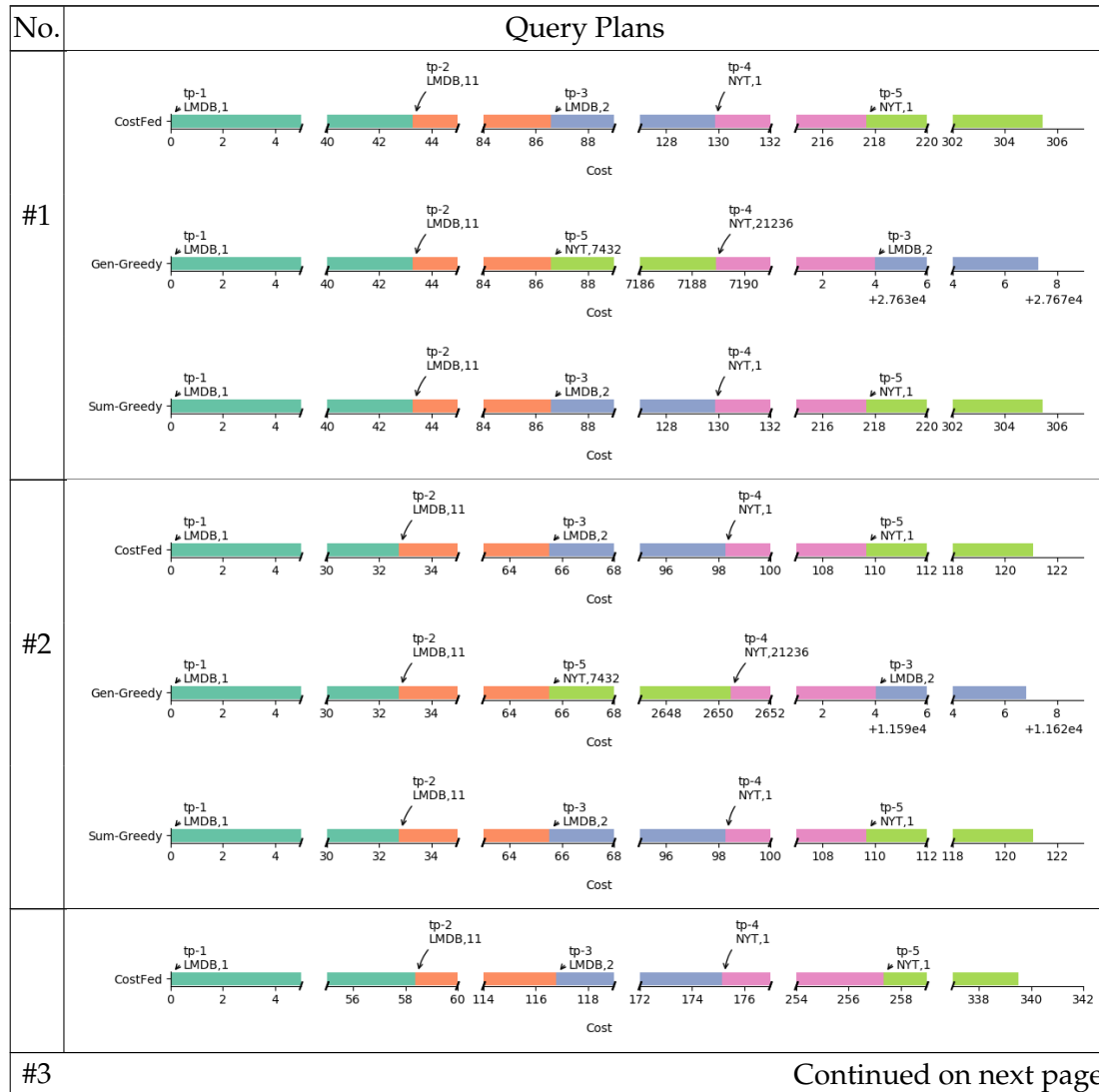


TABLE B.14: Query Plans of Query CD4 at Settings of 10 freemium Pricing Functions



Continued on next page

Table B.14 – continued from previous page

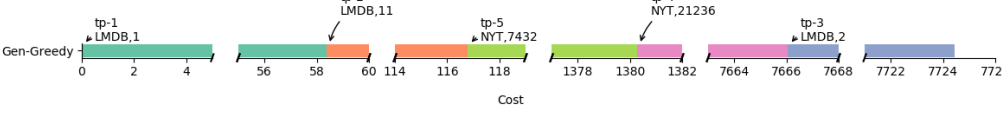
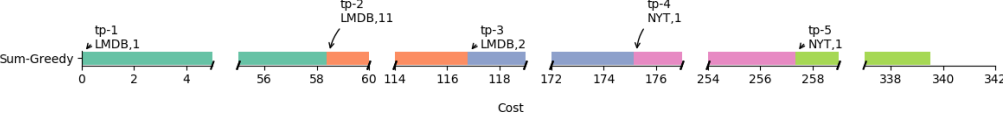
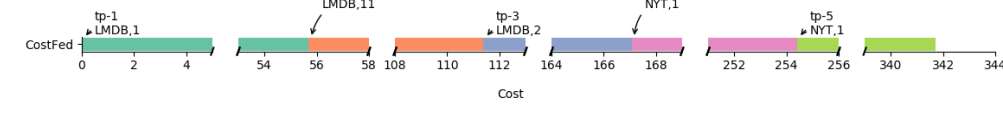
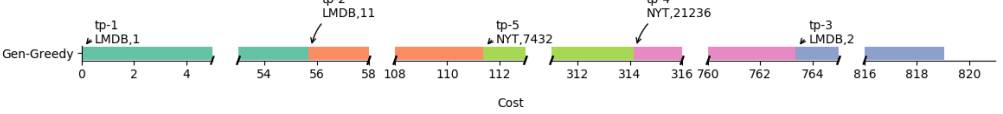
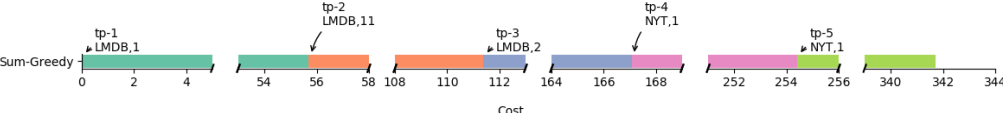
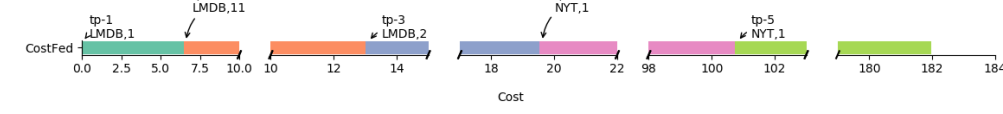
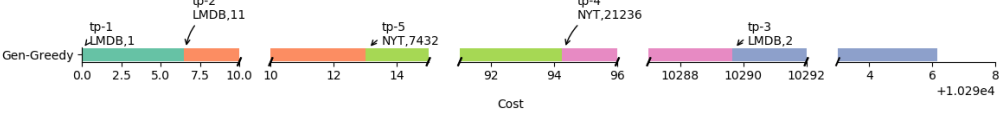
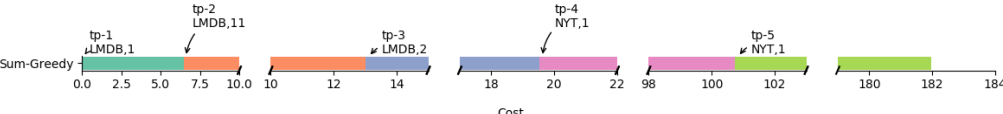
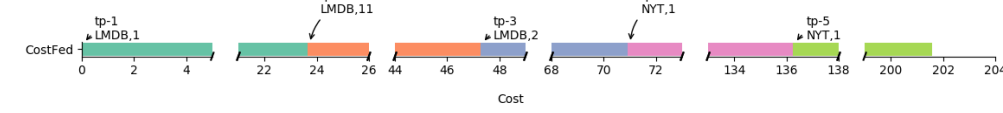
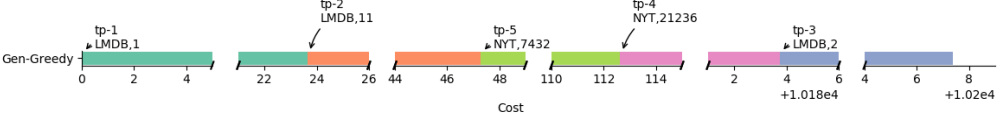
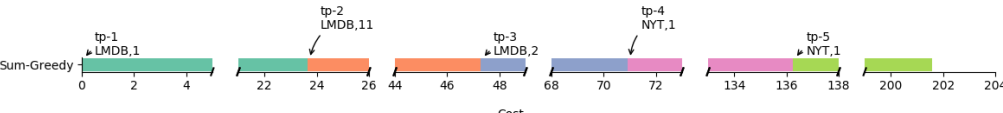
No.	Query Plans
	 <p>Gen-Greedy</p> <p>Cost</p>  <p>Sum-Greedy</p> <p>Cost</p>
#4	 <p>CostFed</p> <p>Cost</p>  <p>Gen-Greedy</p> <p>Cost</p>  <p>Sum-Greedy</p> <p>Cost</p>
#5	 <p>CostFed</p> <p>Cost</p>  <p>Gen-Greedy</p> <p>Cost</p>  <p>Sum-Greedy</p> <p>Cost</p>
#6	 <p>CostFed</p> <p>Cost</p>  <p>Gen-Greedy</p> <p>Cost</p>  <p>Sum-Greedy</p> <p>Cost</p>
Continued on next page	

Table B.14 – continued from previous page

No.	Query Plans
#7	
#8	
#9	
#10	

Continued on next page

Table B.14 – continued from previous page

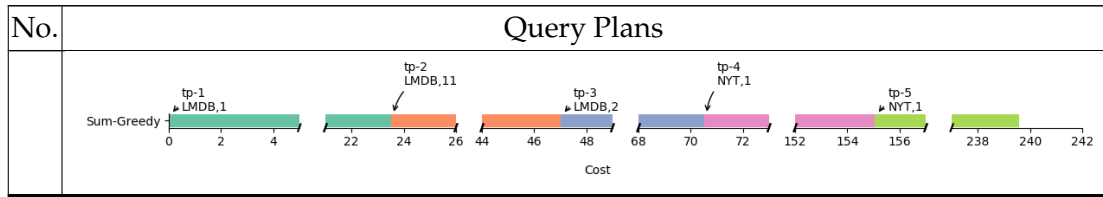


TABLE B.15: Query Plans of Query CD4 at Settings of 10 per Pricing Functions

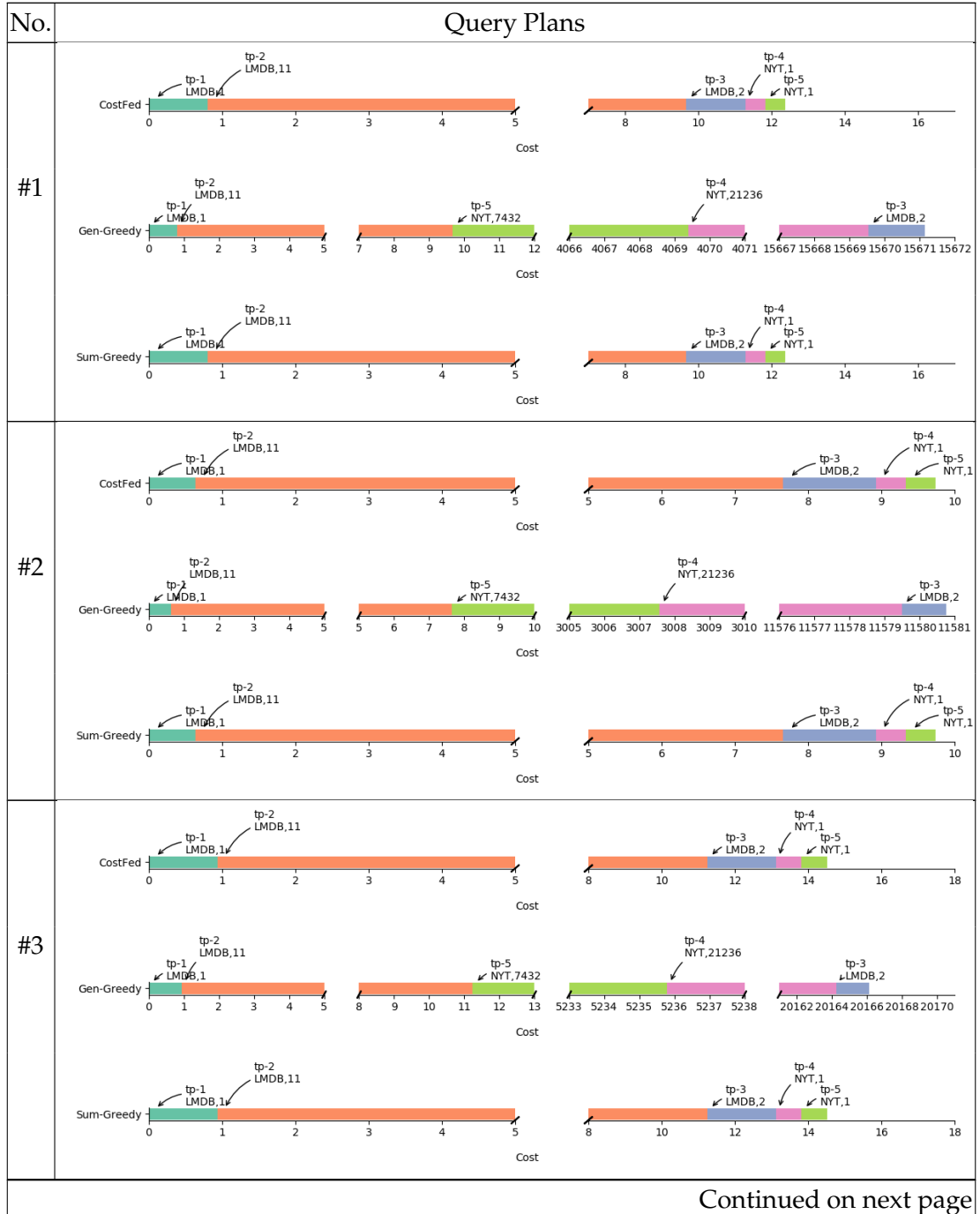
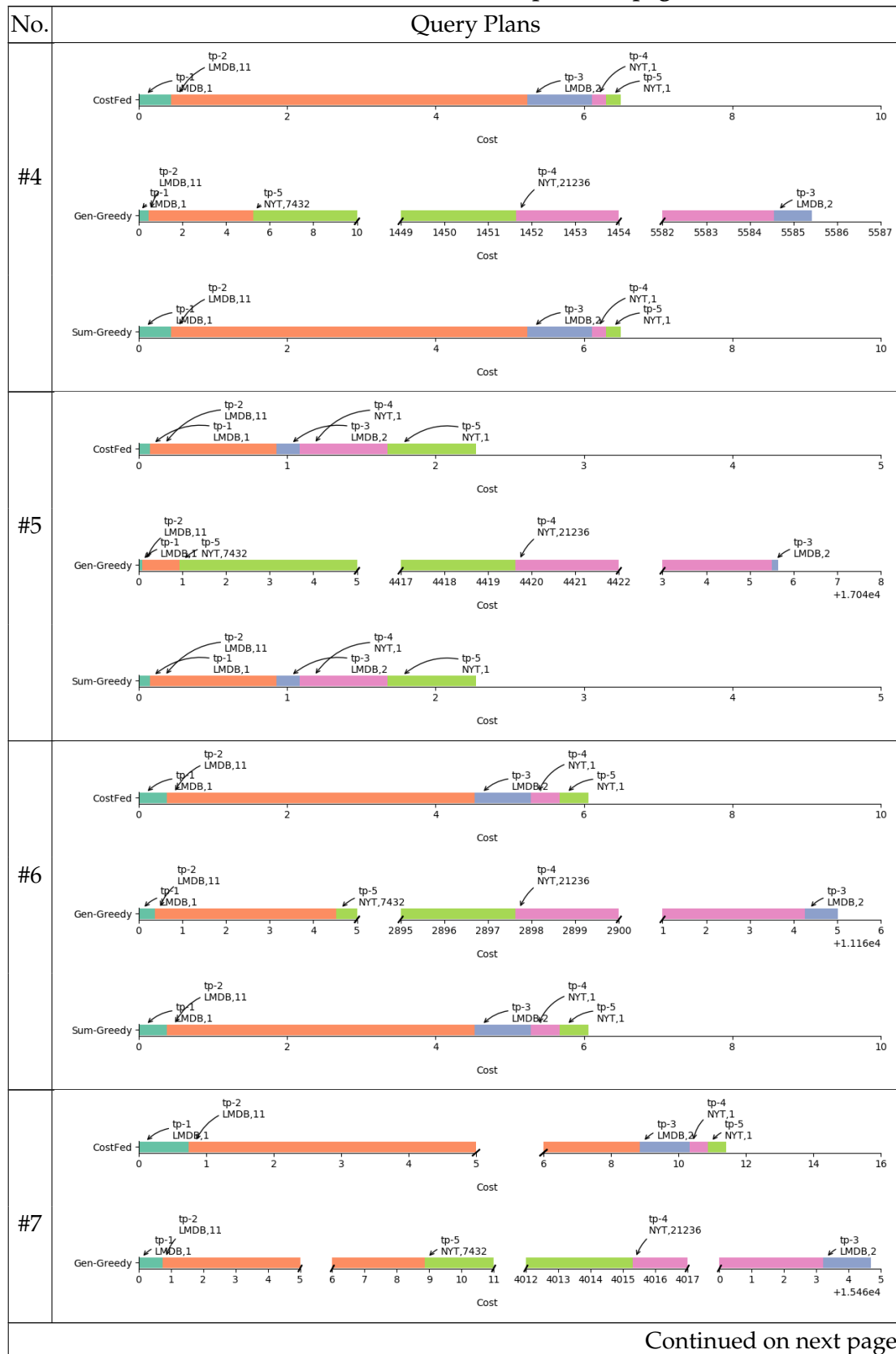


Table B.15 – continued from previous page



Continued on next page

Table B.15 – continued from previous page

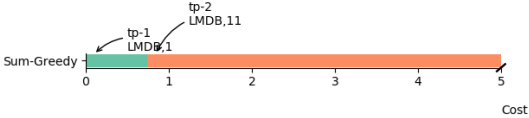
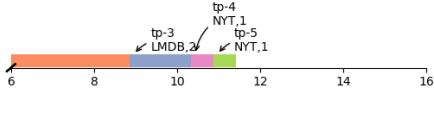
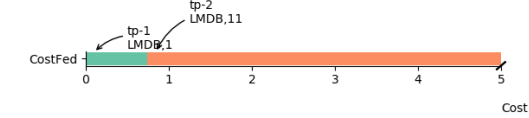
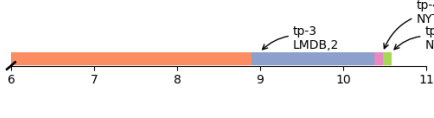
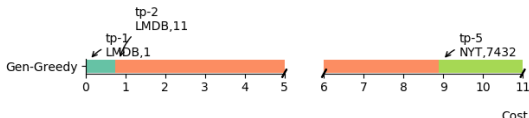
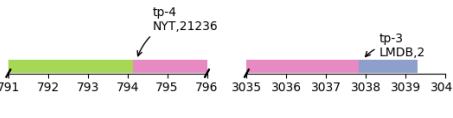
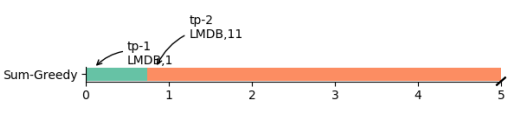
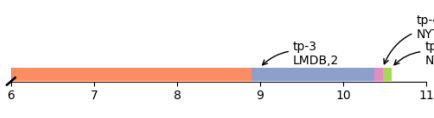
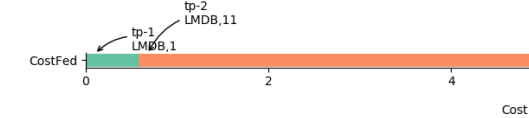
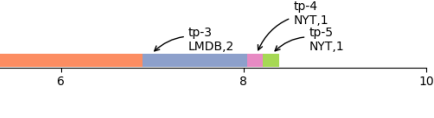
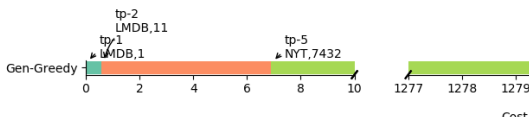
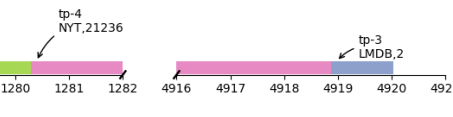
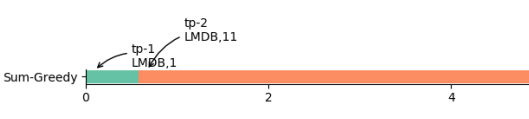
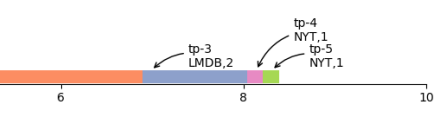
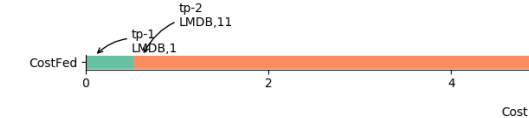
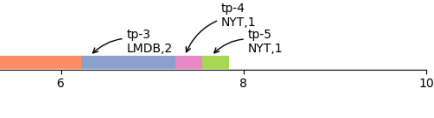
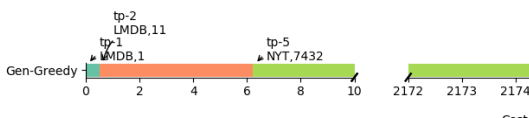
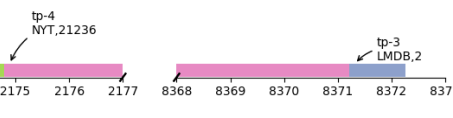
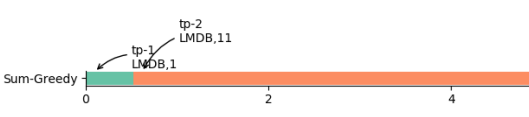
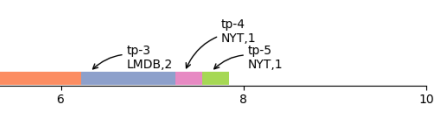
No.	Query Plans
	 
#8	     
#9	     
#10	     

TABLE B.16: Query Plans of Query CD5 at Settings of 10 Flat Pricing Functions

No.	Query Plans
#1	<p>CostFed</p>
	<p>Gen-Greedy</p>
	<p>Sum-Greedy</p>
#2	<p>CostFed</p>
	<p>Gen-Greedy</p>
	<p>Sum-Greedy</p>
#3	<p>CostFed</p>
	<p>Gen-Greedy</p>
	<p>Sum-Greedy</p>
#4	<p>CostFed</p>
	<p>Gen-Greedy</p>

Continued on next page

Table B.16 – continued from previous page

No.	Query Plans
#5	
#6	
#7	
#8	Continued on next page

Table B.16 – continued from previous page

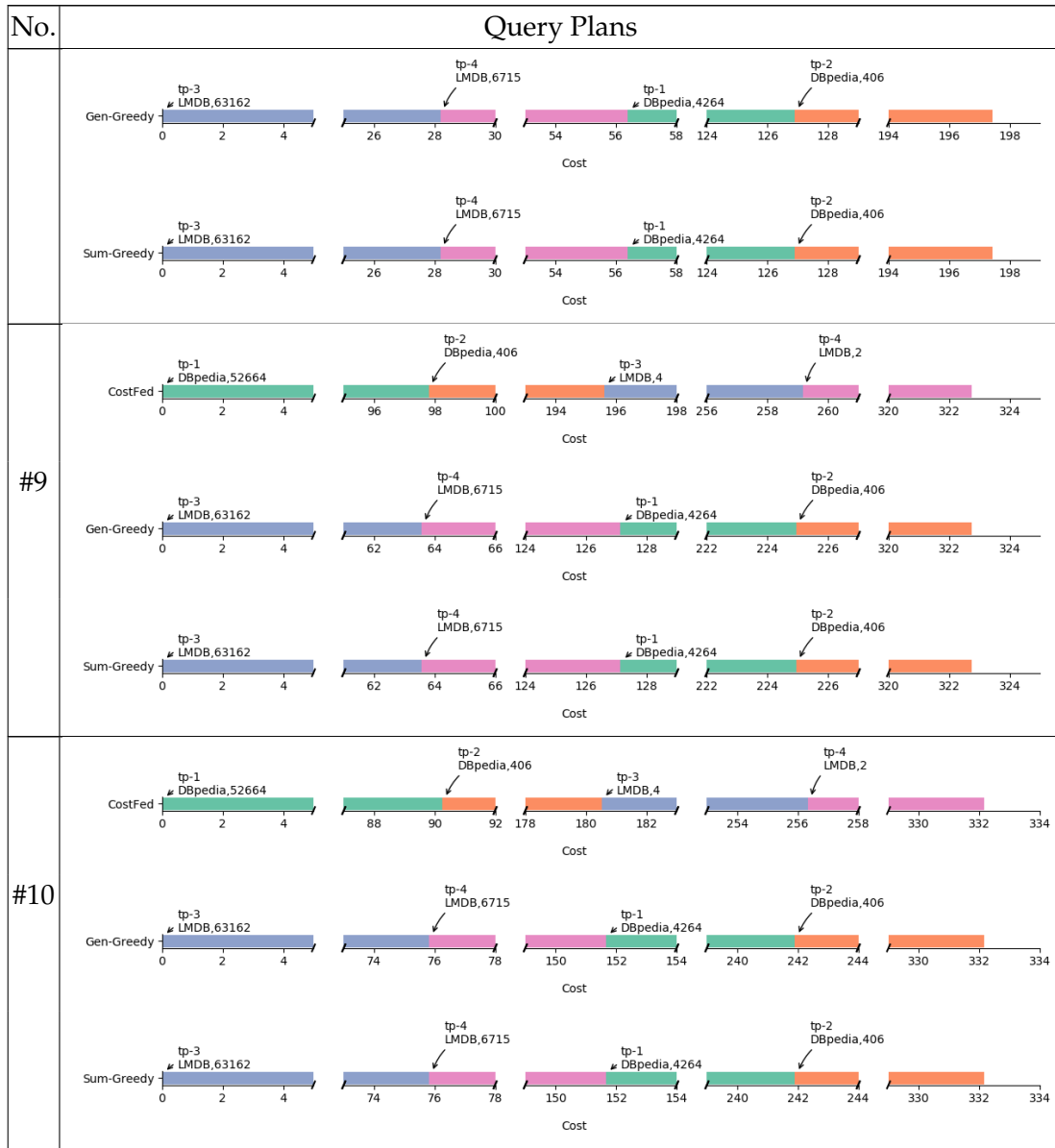
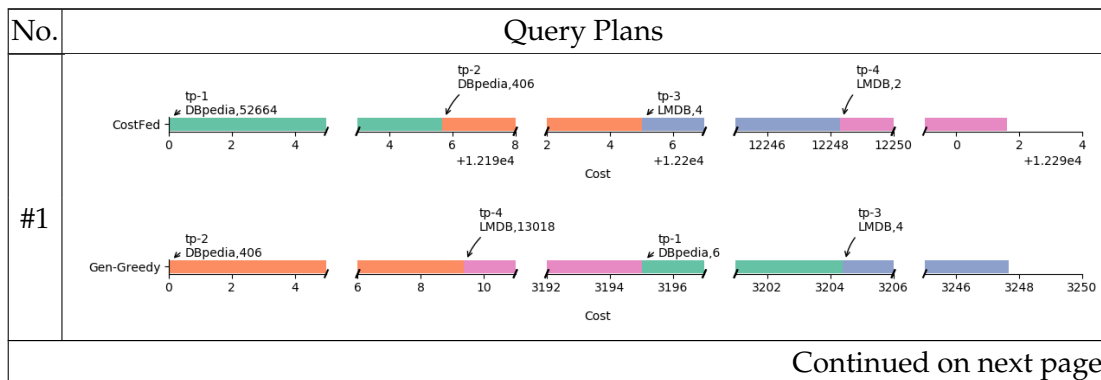


TABLE B.17: Query Plans of Query CD5 at Settings of 10 freemium Pricing Functions

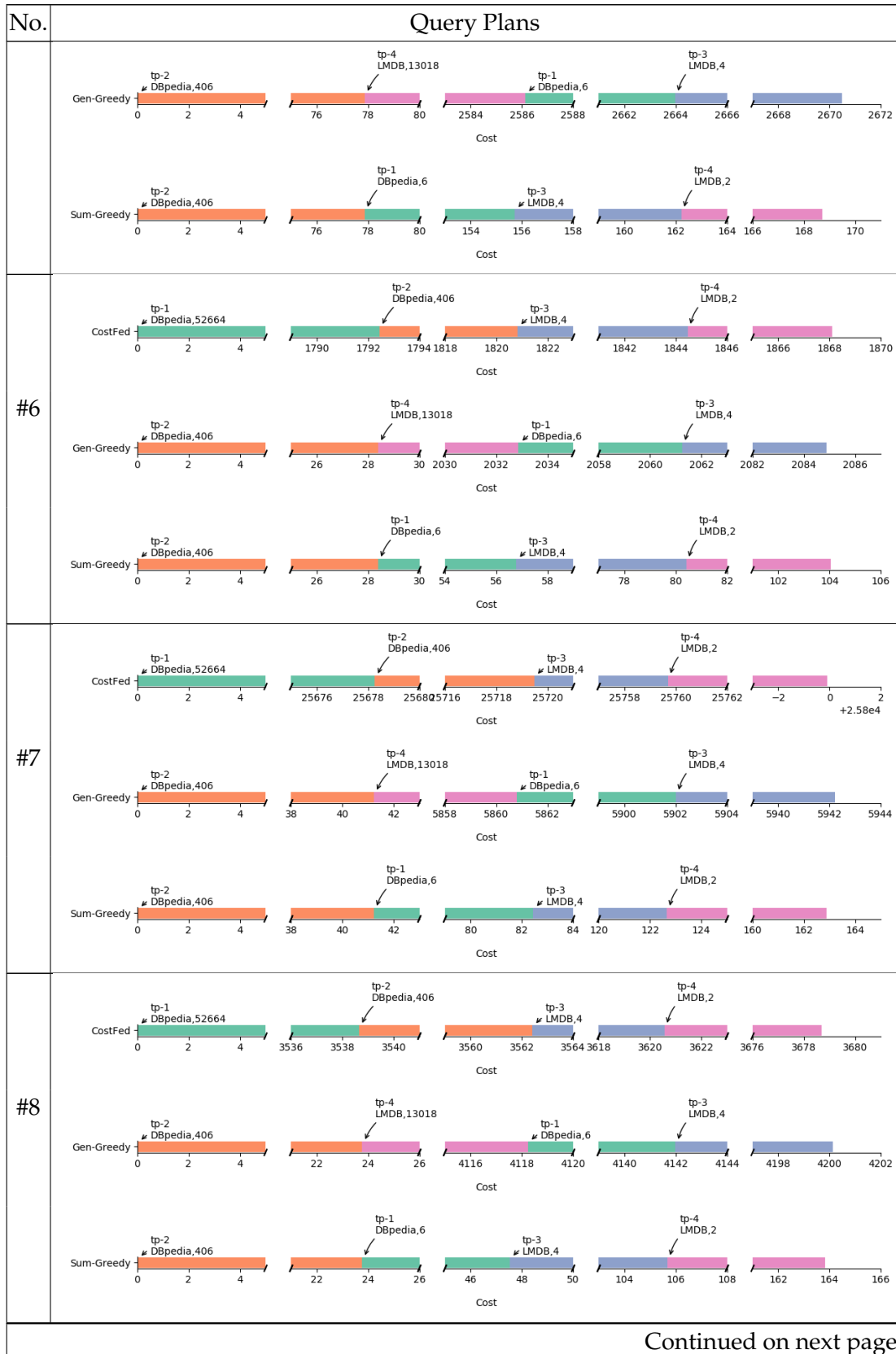


Continued on next page

Table B.17 – continued from previous page

No.	Query Plans
#2	
#3	
#4	
#5	Continued on next page

Table B.17 – continued from previous page



Continued on next page

Table B.17 – continued from previous page

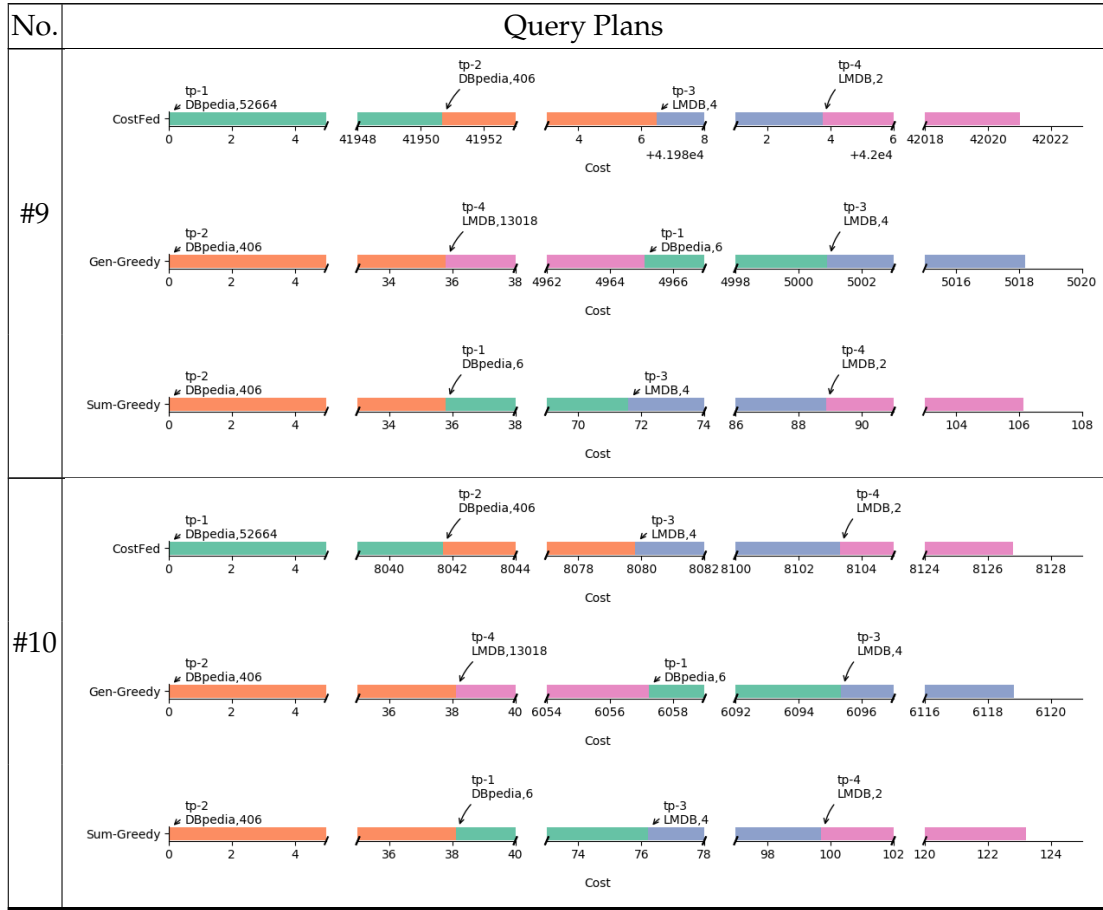
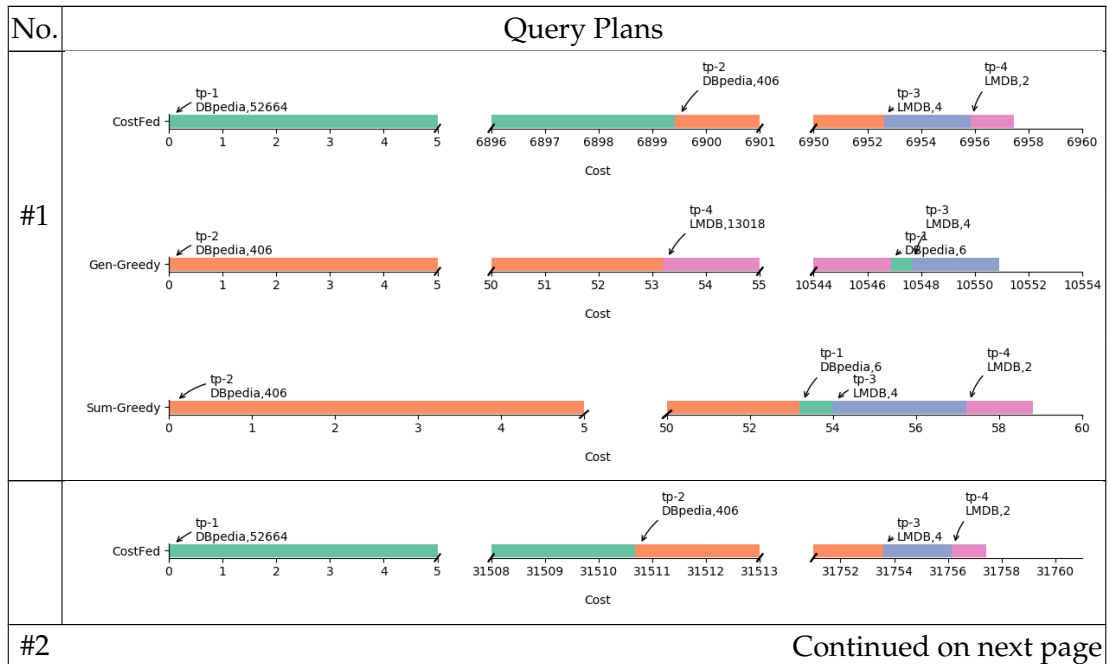


TABLE B.18: Query Plans of Query CD5 at Settings of 10 per Pricing Functions



Continued on next page

Table B.18 – continued from previous page

No.	Query Plans
#3	
#4	
#5	
Continued on next page	

Table B.18 – continued from previous page

No.	Query Plans
#6	<p>CostFed: tp-1 DBpedia,52664 (0-5), tp-2 DBpedia,406 (2-7), Cost: +3.835e4</p> <p>Gen-Greedy: tp-2 DBpedia,406 (0-5), tp-4 LMDB,13018 (293-298), Cost: 293-298</p> <p>Sum-Greedy: tp-2 DBpedia,406 (0-5), tp-1 DBpedia,6 (294-296), Cost: 294-302</p>
#7	<p>CostFed: tp-1 DBpedia,52664 (0-5), tp-2 DBpedia,406 (47975-47980), Cost: 47975-47980</p> <p>Gen-Greedy: tp-2 DBpedia,406 (0-5), tp-4 LMDB,13018 (367-372), Cost: 367-372</p> <p>Sum-Greedy: tp-2 DBpedia,406 (0-5), tp-1 DBpedia,6 (367-372), Cost: 367-372</p>
#8	<p>CostFed: tp-1 DBpedia,52664 (0-5), tp-2 DBpedia,406 (4-9), Cost: +2.553e4</p> <p>Gen-Greedy: tp-2 DBpedia,406 (0-5), tp-4 LMDB,13018 (194-199), Cost: 194-199</p> <p>Sum-Greedy: tp-2 DBpedia,406 (0-5), tp-1 DBpedia,6 (194-196), Cost: 194-208</p>
#9	<p>CostFed: tp-1 DBpedia,52664 (0-5), tp-2 DBpedia,406 (2-7), Cost: +4.53e4</p> <p>Gen-Greedy: tp-2 DBpedia,406 (0-5), tp-4 LMDB,13018 (346-351), Cost: 346-351</p>

Continued on next page

Table B.18 – continued from previous page

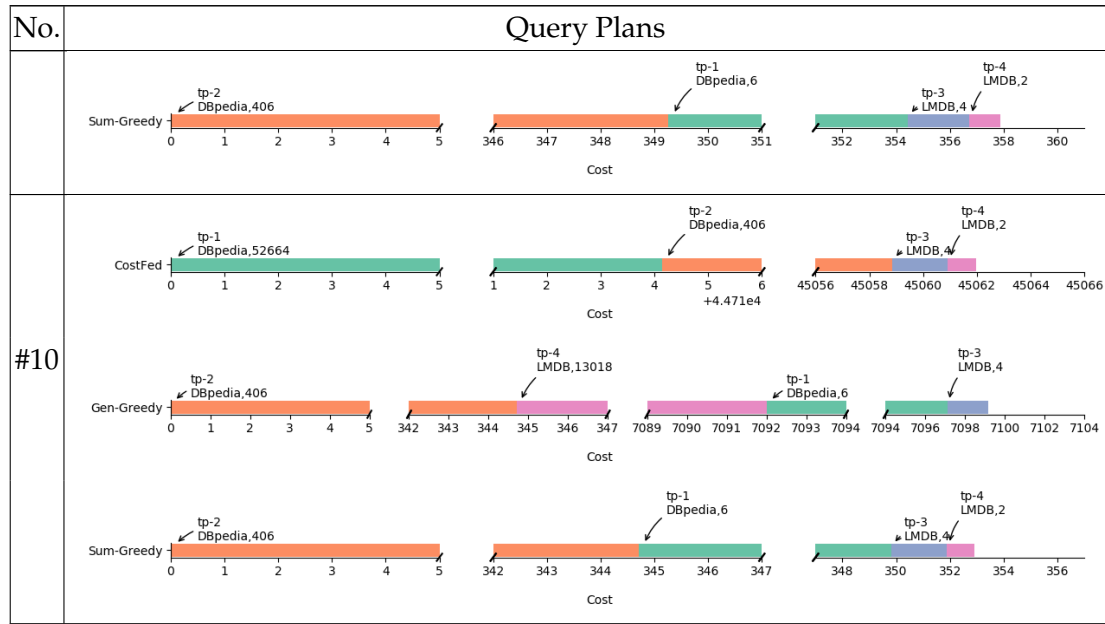
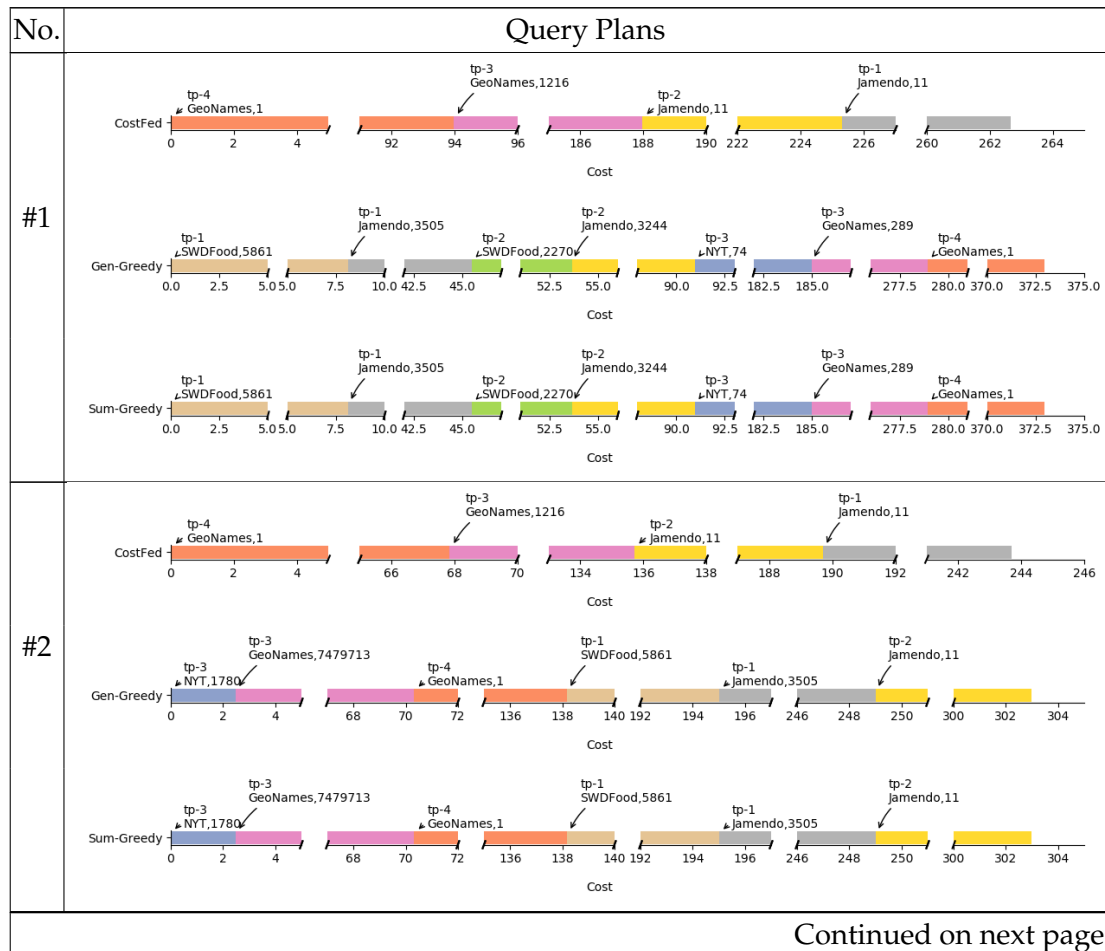


TABLE B.19: Query Plans of Query CD6 at Settings of 10 Flat Pricing Functions



Continued on next page

Table B.19 – continued from previous page

No.	Query Plans
#3	<p>CostFed: tp-4 GeoNames,1 (0.0 to 15.0), tp-3 GeoNames,1216 (96 to 101), tp-2 jamendo,11 (184 to 189).</p> <p>Gen-Greedy: tp-3 NYT,1780 (0 to 4), tp-3 GeoNames,7479713 (22 to 30), tp-4 GeoNames,1 (32 to 36), tp-1 SWDFood,5861 (110 to 114), tp-1 jamendo,3505 (198 to 202), tp-2 jamendo,11 (286 to 290).</p> <p>Sum-Greedy: tp-3 NYT,1780 (0 to 4), tp-3 GeoNames,7479713 (22 to 30), tp-4 GeoNames,1 (32 to 36), tp-1 SWDFood,5861 (110 to 114), tp-1 jamendo,3505 (198 to 202), tp-2 jamendo,11 (286 to 290).</p>
#4	<p>CostFed: tp-4 GeoNames,1 (0 to 4), tp-3 GeoNames,1216 (62 to 66), tp-2 jamendo,11 (126 to 130), tp-1 jamendo,11 (178 to 182).</p> <p>Gen-Greedy: tp-1 SWDFood,5861 (0.0 to 5.0), tp-1 jamendo,3505 (27.5 to 30.0), tp-2 SWDFood,2270 (80.0 to 82.5), tp-2 jamendo,3244 (110.0 to 112.5), tp-3 NYT,74 (160.0 to 162.5), tp-3 GeoNames,289 (250.0 to 252.5), tp-4 GeoNames,1 (315.0 to 380.0).</p> <p>Sum-Greedy: tp-1 SWDFood,5861 (0.0 to 5.0), tp-1 jamendo,3505 (27.5 to 30.0), tp-2 SWDFood,2270 (80.0 to 82.5), tp-2 jamendo,3244 (110.0 to 112.5), tp-3 NYT,74 (160.0 to 162.5), tp-3 GeoNames,289 (250.0 to 252.5), tp-4 GeoNames,1 (315.0 to 380.0).</p>
#5	<p>CostFed: tp-4 GeoNames,1 (0 to 4), tp-3 GeoNames,1216 (16 to 20), tp-2 jamendo,11 (34 to 38), tp-1 jamendo,11 (134 to 138).</p> <p>Gen-Greedy: tp-3 NYT,1780 (0 to 4), tp-3 GeoNames,7479713 (18 to 22), tp-4 GeoNames,1 (36 to 40), tp-1 SWDFood,5861 (108 to 112), tp-1 jamendo,3505 (208 to 212), tp-2 jamendo,11 (308 to 312).</p> <p>Sum-Greedy: tp-3 NYT,1780 (0 to 4), tp-3 GeoNames,7479713 (18 to 22), tp-4 GeoNames,1 (36 to 40), tp-1 SWDFood,5861 (108 to 112), tp-1 jamendo,3505 (208 to 212), tp-2 jamendo,11 (308 to 312).</p>
#6	<p>CostFed: tp-4 GeoNames,1 (0 to 4), tp-3 GeoNames,1216 (56 to 60), tp-2 jamendo,11 (116 to 120), tp-1 jamendo,11 (154 to 158).</p> <p>Gen-Greedy: tp-1 SWDFood,5861 (0.0 to 5.0), tp-1 jamendo,3505 (62.5 to 65.0), tp-2 SWDFood,2270 (100.0 to 102.5), tp-2 jamendo,3244 (167.5 to 170.0), tp-3 NYT,74 (205.0 to 207.5), tp-3 GeoNames,289 (300.0 to 302.5), tp-4 GeoNames,1 (360.0 to 420.0).</p>

Continued on next page

Table B.19 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#8	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#9	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#10	<p>CostFed</p> <p>Cost</p>

Continued on next page

Table B.19 – continued from previous page

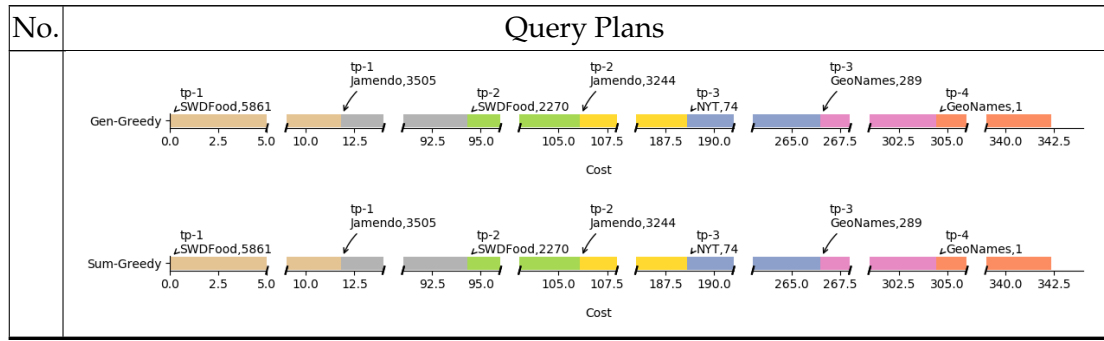
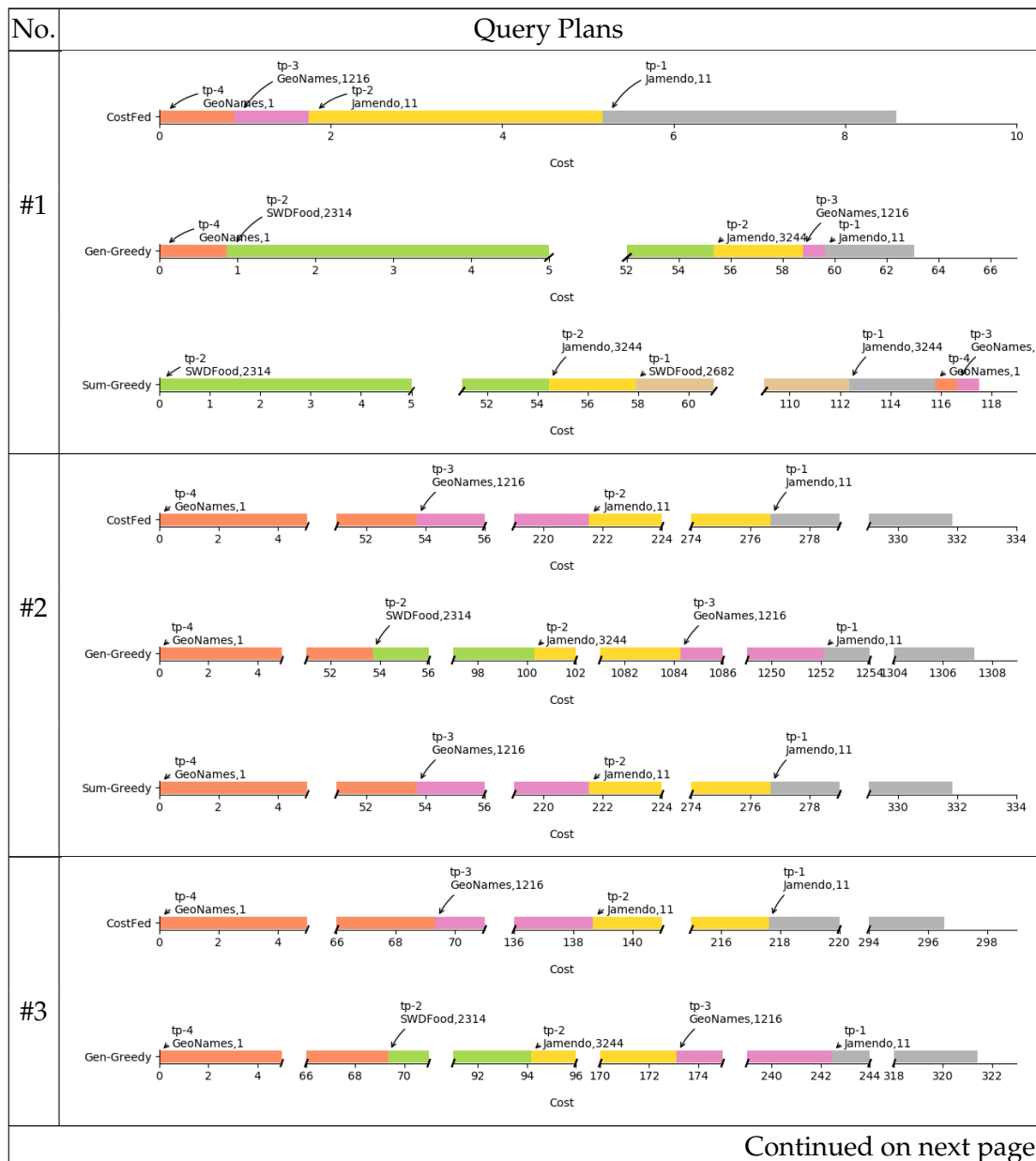


TABLE B.20: Query Plans of Query CD6 at Settings of 10 freemium Pricing Functions



Continued on next page

Table B.20 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#4	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#5	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#6	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Cost</p>

Continued on next page

Table B.20 – continued from previous page

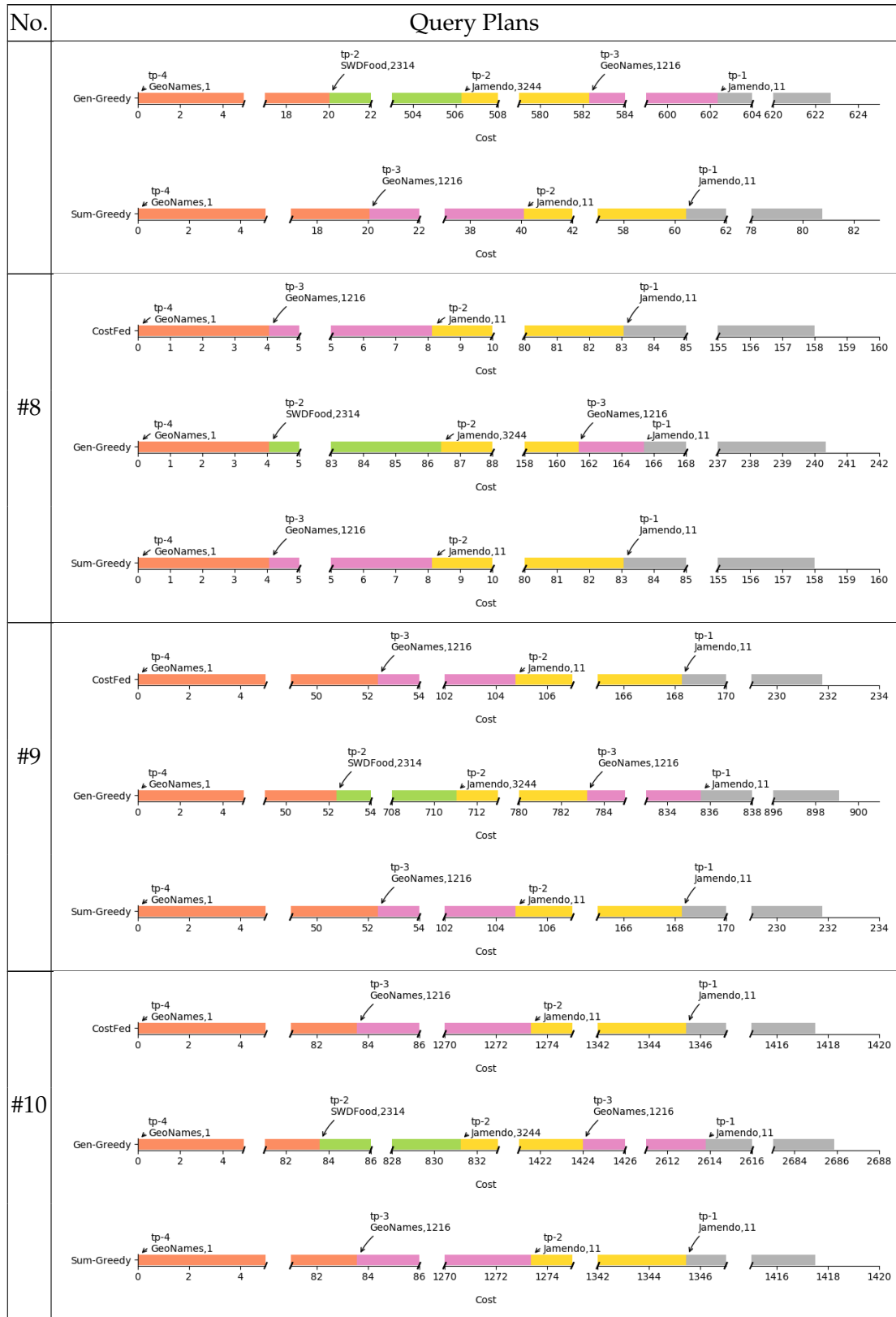


TABLE B.21: Query Plans of Query CD6 at Settings of 10 per Pricing Functions

No.	Query Plans
#1	
#2	
#3	
#4	

Continued on next page

Table B.21 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#5	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#6	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#8	<p>CostFed</p> <p>Cost</p>

Continued on next page

Table B.21 – continued from previous page

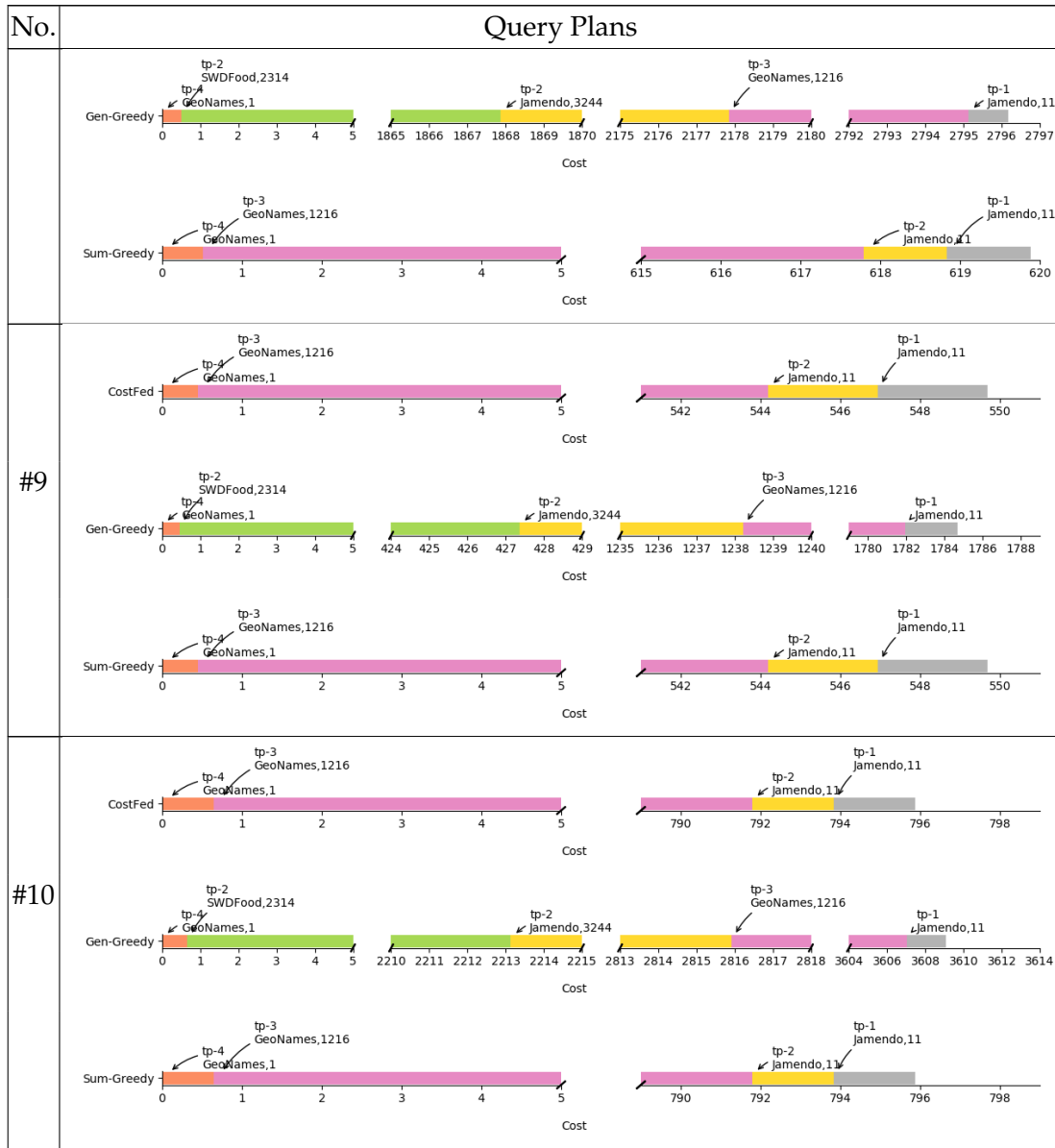
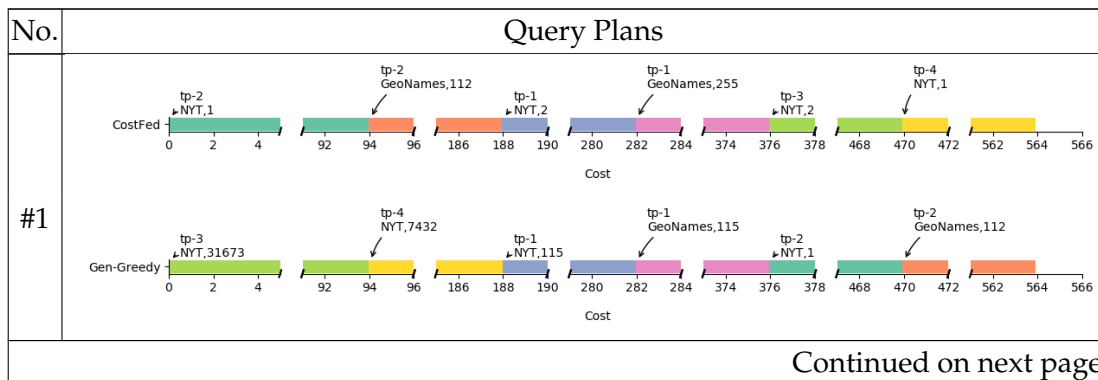


TABLE B.22: Query Plans of Query CD7 at Settings of 10 Flat Pricing Functions



Continued on next page

Table B.22 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#2	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#3	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#4	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#5	<p>CostFed</p> <p>Cost</p>

Continued on next page

Table B.22 – continued from previous page

No.	Query Plans
	<p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#6	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#8	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>

Continued on next page

Table B.22 – continued from previous page

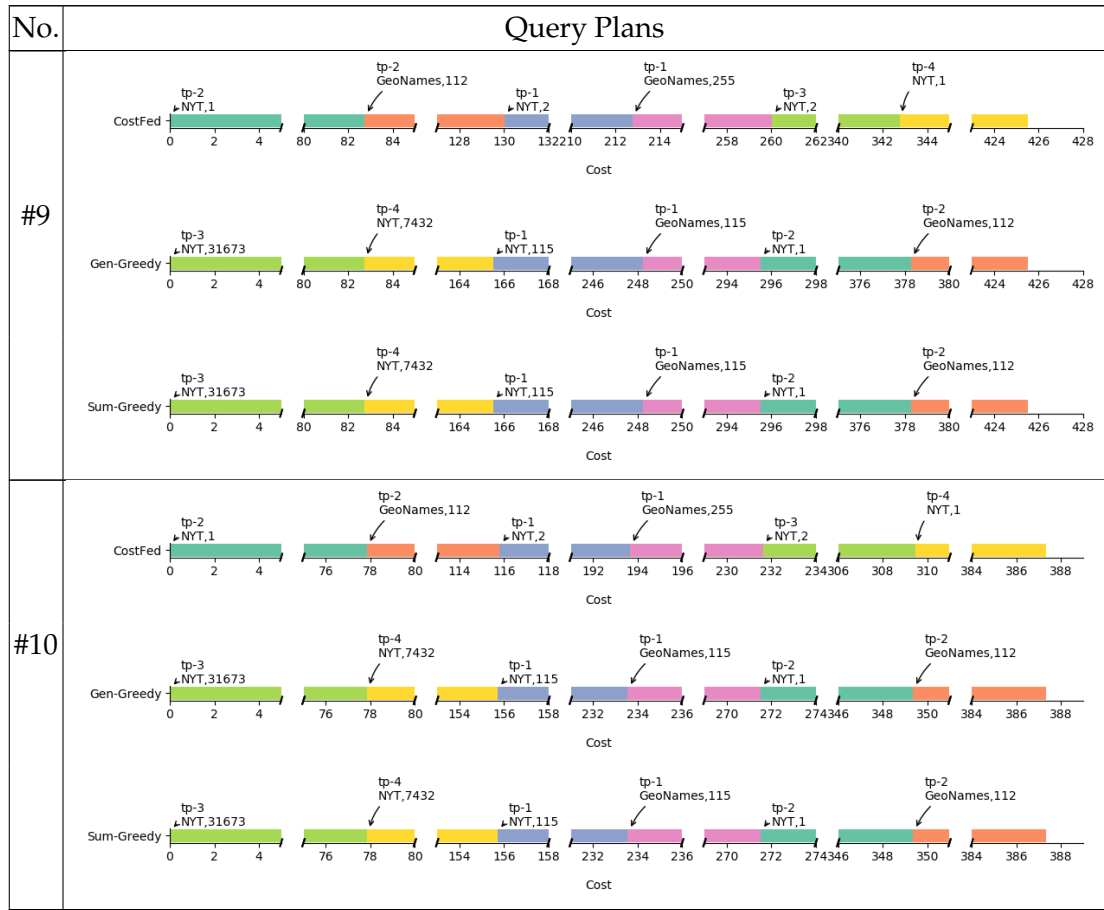
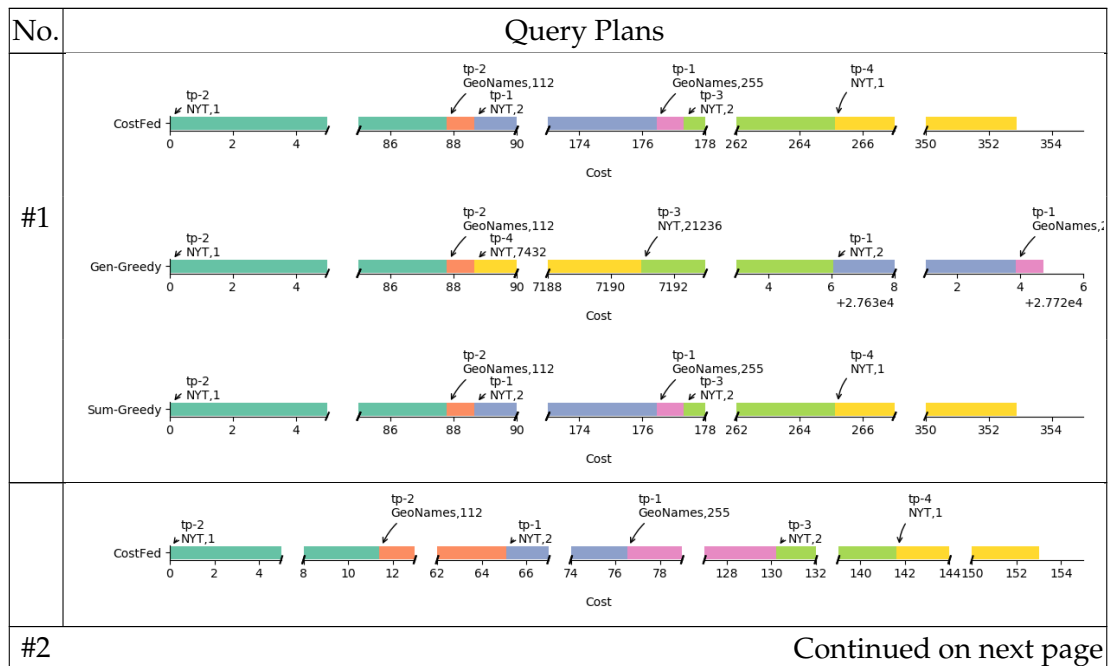


TABLE B.23: Query Plans of Query CD7 at Settings of 10 freemium Pricing Functions



Continued on next page

Table B.23 – continued from previous page

No.	Query Plans
	<p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#3	<p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#4	<p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#5	<p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>

Continued on next page

Table B.23 – continued from previous page

No.	Query Plans
#6	<p>CostFed: tp-2 NYT,1 (0-4), tp-2 GeoNames,112 (62-66), tp-1 NYT,2 (86-90), tp-1 GeoNames,255 (150-154), tp-3 NYT,2 (172-176), tp-4 NYT,1 (238-242), 304-308.</p> <p>Gen-Greedy: tp-2 NYT,1 (0-4), tp-2 GeoNames,112 (62-66), tp-4 NYT,7432 (86-90), tp-3 NYT,21236 (150-154), tp-1 NYT,2 (2-4), tp-1 GeoNames,255 (610286-10290), +1.022e4, +1.031e4.</p> <p>Sum-Greedy: tp-4 NYT,7432 (0-4), tp-2 NYT,1 (62-66), tp-2 GeoNames,112 (128-130), tp-1 NYT,2 (150-154), tp-1 GeoNames,255 (216-218), tp-3 NYT,2 (238-242), 304-308.</p>
#7	<p>CostFed: tp-2 NYT,1 (0-4), tp-2 GeoNames,112 (36-40), tp-1 NYT,2 (56-60), tp-1 GeoNames,255 (94-96), tp-3 NYT,2 (114-118), tp-4 NYT,1 (154-156), 158-196.</p> <p>Gen-Greedy: tp-2 NYT,1 (0-4), tp-2 GeoNames,112 (36-40), tp-4 NYT,7432 (56-60), tp-3 NYT,21236 (1816-1820), tp-1 NYT,2 (7240-7242), tp-1 GeoNames,255 (7248-7280), 7298-7302.</p> <p>Sum-Greedy: tp-2 NYT,1 (0-4), tp-2 GeoNames,112 (36-40), tp-1 NYT,2 (56-60), tp-1 GeoNames,255 (94-96), tp-3 NYT,2 (114-118), tp-4 NYT,1 (154-156), 158-196.</p>
#8	<p>CostFed: tp-2 NYT,1 (0-4), tp-2 GeoNames,112 (72.5-75.0), tp-1 NYT,2 (77.5-80.0), tp-1 GeoNames,255 (152.5-155.0), tp-3 NYT,2 (157.5-160.0), 232-236, 306-310.</p> <p>Gen-Greedy: tp-2 NYT,1 (0-4), tp-2 GeoNames,112 (72.5-75.0), tp-4 NYT,7432 (77.5-80.0), tp-3 NYT,21236 (152-154), tp-1 NYT,2 (2-4), tp-1 GeoNames,255 (12587.12590.12592.12595.0), +1.251e4.</p> <p>Sum-Greedy: tp-4 NYT,7432 (0-4), tp-2 NYT,1 (72-74), tp-2 GeoNames,112 (150.0-152.5), tp-1 NYT,2 (155.0-157.5), tp-1 GeoNames,255 (227.5-230.0), tp-3 NYT,2 (232.5-235.0), 306-310.</p>
#9	<p>CostFed: tp-2 NYT,1 (0-4), tp-2 GeoNames,112 (72-74), tp-1 NYT,2 (124-126), tp-1 GeoNames,255 (198-200), tp-3 NYT,2 (202-250), 252-254, 324-326, 328-398, 400-402.</p> <p>Gen-Greedy: tp-2 NYT,1 (0-4), tp-2 GeoNames,112 (72-74), tp-4 NYT,7432 (124-126), tp-3 NYT,21236 (128-1120), tp-1 NYT,2 (5768-5770), tp-1 GeoNames,255 (5842-5844), 5846-5894, 5896-5898.</p>

Continued on next page

Table B.23 – continued from previous page

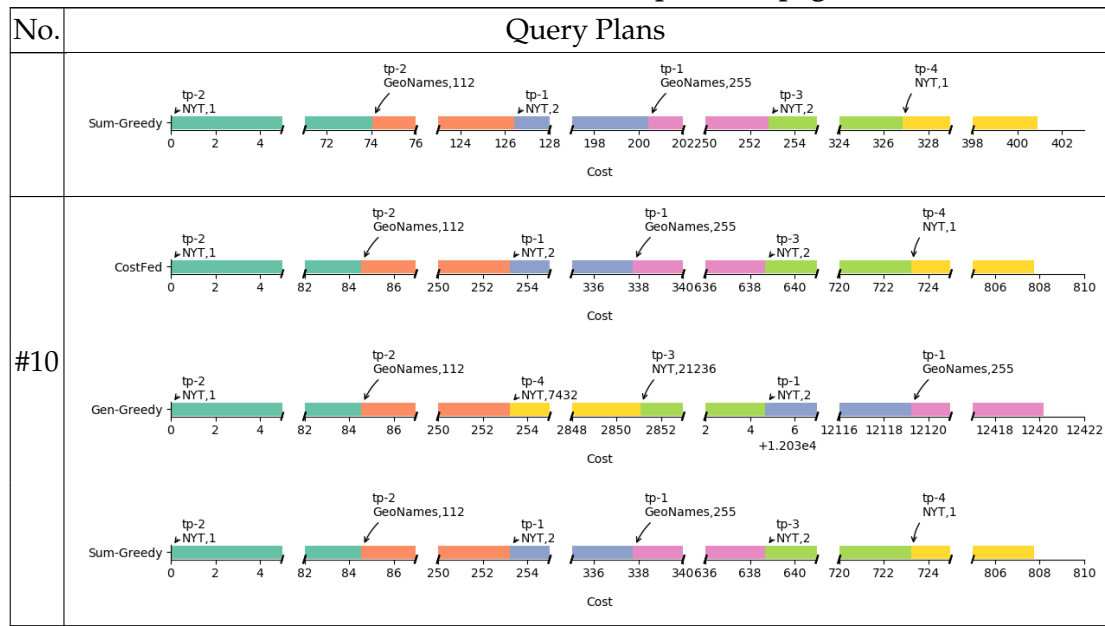
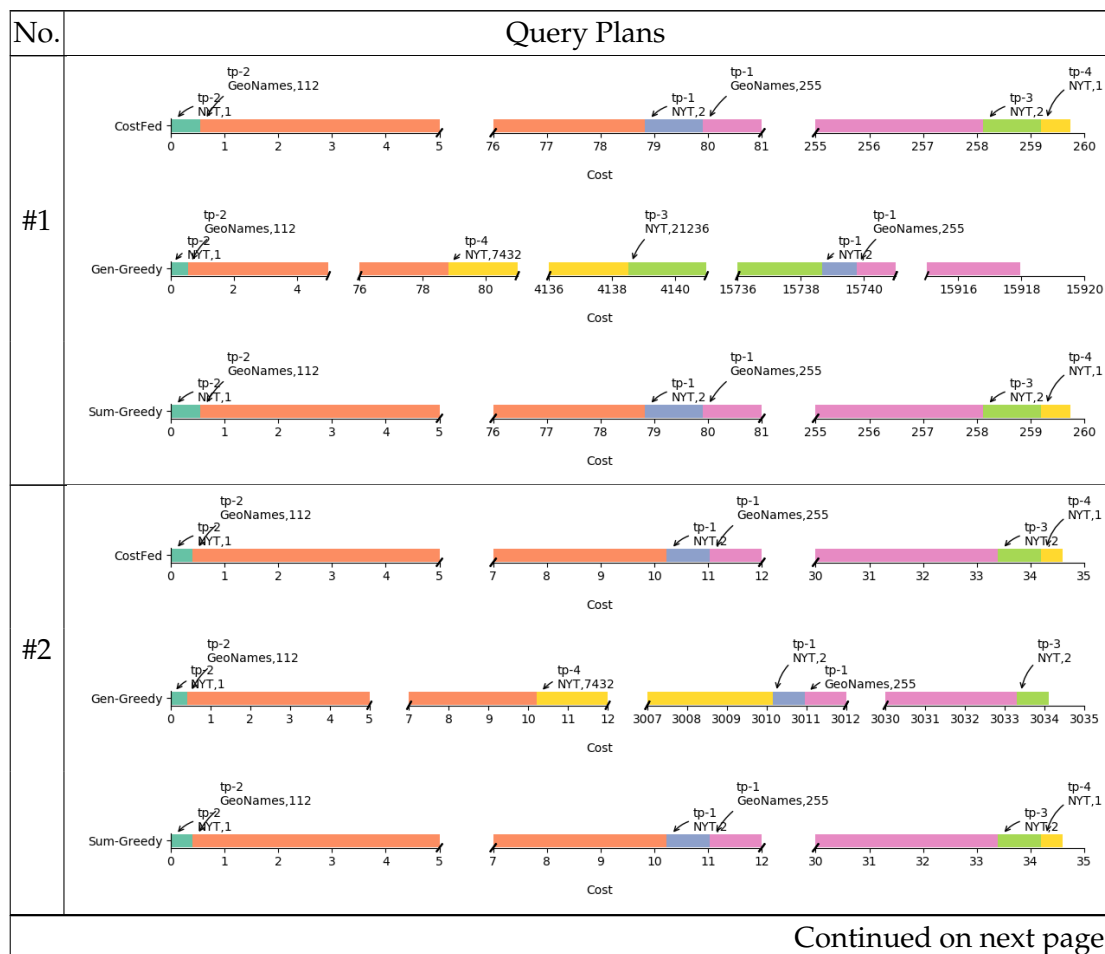


TABLE B.24: Query Plans of Query CD7 at Settings of 10 per Pricing Functions



Continued on next page

Table B.24 – continued from previous page

No.	Query Plans
#3	<p>CostFed: tp-2 GeoNames,112 (0-5), tp-1 NYT,2 (97-102), tp-1 GeoNames,255 (324-332)</p> <p>Gen-Greedy: tp-2 GeoNames,112 (0-5), tp-4 NYT,7432 (98-102), tp-3 NYT,21236 (5322-5326), tp-1 NYT,2 (0-2), tp-1 GeoNames,255 (20476-20480)</p> <p>Sum-Greedy: tp-2 GeoNames,112 (0-5), tp-1 NYT,2 (97-102), tp-1 GeoNames,255 (324-332)</p>
#4	<p>CostFed: tp-2 GeoNames,112 (0-10), tp-1 NYT,2 (16-21), tp-1 GeoNames,255 (1449-1468)</p> <p>Gen-Greedy: tp-2 GeoNames,112 (0-10), tp-4 NYT,7432 (1449-1452), tp-3 NYT,21236 (1452-1453), tp-1 NYT,2 (1453-1454), tp-1 GeoNames,255 (1463-1466)</p> <p>Sum-Greedy: tp-2 GeoNames,112 (0-10), tp-1 NYT,2 (16-21), tp-1 GeoNames,255 (1449-1468)</p>
#5	<p>CostFed: tp-2 GeoNames,112 (0-5), tp-1 NYT,2 (38-43), tp-1 GeoNames,255 (131-136)</p> <p>Gen-Greedy: tp-2 GeoNames,112 (0-5), tp-4 NYT,7432 (38-42), tp-3 NYT,21236 (4456-4460), tp-1 NYT,2 (2-6), tp-1 GeoNames,255 (17176-17180)</p> <p>Sum-Greedy: tp-2 GeoNames,112 (0-5), tp-1 NYT,2 (38-43), tp-1 GeoNames,255 (131-136)</p>
#6	<p>CostFed: tp-2 GeoNames,112 (0-5), tp-1 NYT,2 (53-58), tp-1 GeoNames,255 (179-184)</p> <p>Gen-Greedy: tp-2 GeoNames,112 (0-5), tp-4 NYT,7432 (54-58), tp-3 NYT,21236 (2946-2950), tp-1 NYT,2 (2-6), tp-1 GeoNames,255 (17176-17180)</p> <p>Sum-Greedy: tp-2 GeoNames,112 (0-5), tp-1 NYT,2 (53-58), tp-1 GeoNames,255 (179-184)</p>

Continued on next page

Table B.24 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>tp-2 GeoNames,112 tp-1 GeoNames,255 tp-3 NYT,2</p> <p>Cost: 0 1 2 3 4 5 53 54 55 56 57 58 179 180 181 182 183 184</p>
#7	<p>CostFed</p> <p>tp-2 GeoNames,112 tp-1 GeoNames,255 tp-3 NYT,2</p> <p>Cost: 0 1 2 3 4 5 73 74 75 76 77 78 245 246 247 248 249 250</p> <p>Gen-Greedy</p> <p>tp-2 GeoNames,112 tp-4 NYT,7432 tp-3 NYT,21236 tp-1 GeoNames,255</p> <p>Cost: 0 2 4 74 76 78 4080 4082 4084 15528 15530 15532 0 2 +1.57e4</p> <p>Sum-Greedy</p> <p>tp-2 GeoNames,112 tp-1 GeoNames,255 tp-3 NYT,2</p> <p>Cost: 0 1 2 3 4 5 73 74 75 76 77 78 245 246 247 248 249 250</p>
#8	<p>CostFed</p> <p>tp-2 GeoNames,112 tp-1 GeoNames,255 tp-3 NYT,2</p> <p>Cost: 0 1 2 3 4 5 54 55 56 57 58 59 184 185 186 187 188 189</p> <p>Gen-Greedy</p> <p>tp-2 GeoNames,112 tp-4 NYT,7432 tp-3 NYT,21236 tp-1 GeoNames,255</p> <p>Cost: 0 2 4 54 56 58 840 842 844 3084 3086 3088 3214 3216 3218</p> <p>Sum-Greedy</p> <p>tp-2 GeoNames,112 tp-1 GeoNames,255 tp-3 NYT,2</p> <p>Cost: 0 1 2 3 4 5 54 55 56 57 58 59 184 185 186 187 188 189</p>
#9	<p>CostFed</p> <p>tp-2 GeoNames,112 tp-1 GeoNames,255 tp-3 NYT,2</p> <p>Cost: 0 1 2 3 4 5 47 48 49 50 51 52 162 163 164 165 166 167</p> <p>Gen-Greedy</p> <p>tp-2 GeoNames,112 tp-4 NYT,7432 tp-3 NYT,21236 tp-1 GeoNames,255</p> <p>Cost: 0 2 4 48 50 52 1322 1324 1326 4960 4962 4964 5074 5076 5078</p> <p>Sum-Greedy</p> <p>tp-2 GeoNames,112 tp-1 GeoNames,255 tp-3 NYT,2</p> <p>Cost: 0 1 2 3 4 5 47 48 49 50 51 52 162 163 164 165 166 167</p>
	<p>CostFed</p> <p>tp-2 GeoNames,112 tp-1 GeoNames,255 tp-3 NYT,2</p> <p>Cost: 0 1 2 3 4 5 70 71 72 73 74 75 237 238 239 240 241 242</p>
#10	Continued on next page

Table B.24 – continued from previous page

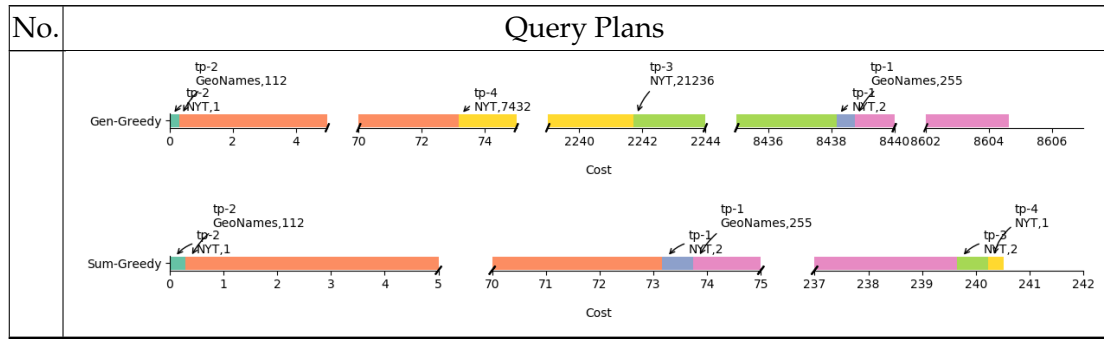
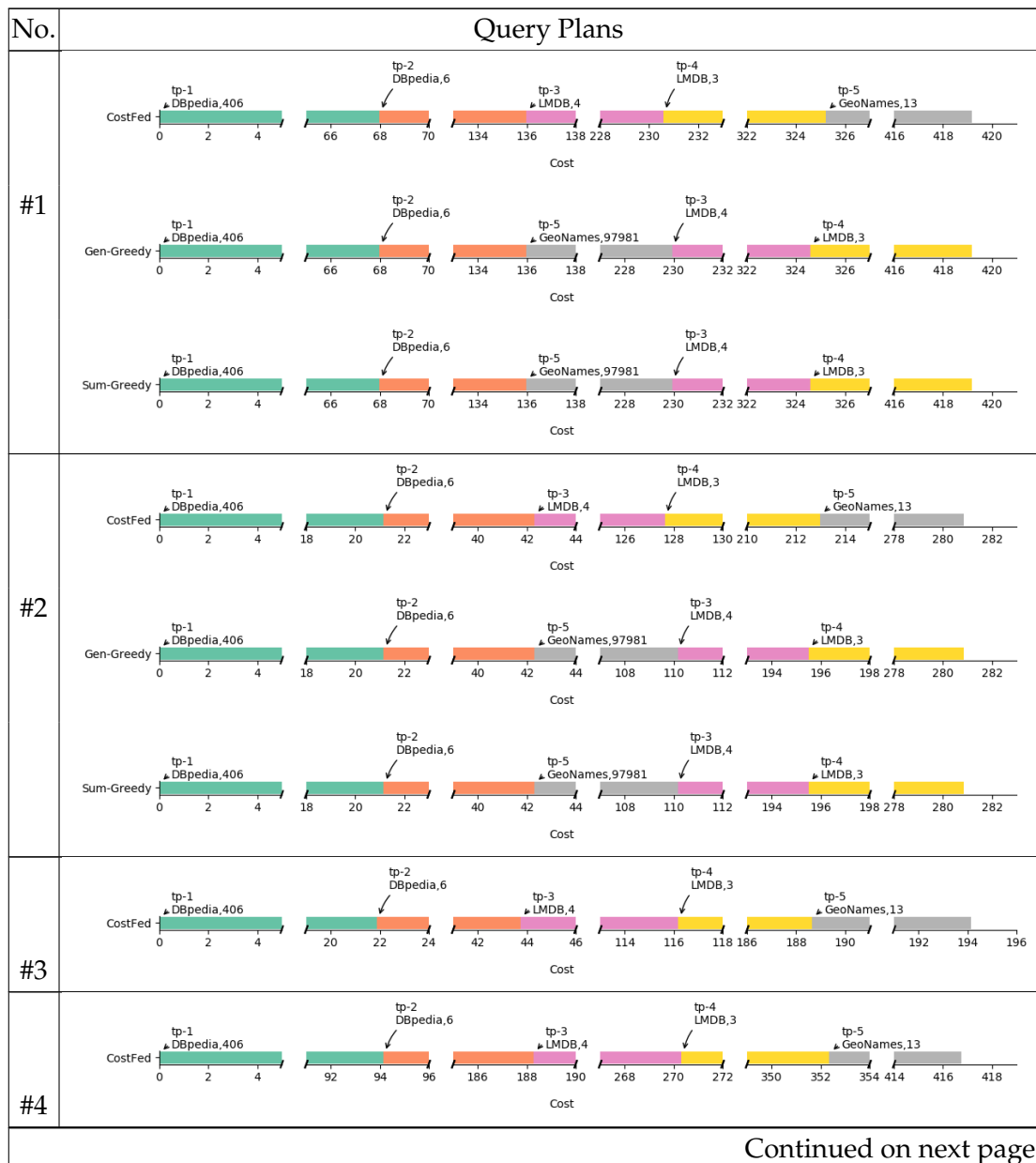


TABLE B.25: Query Plans of Query LD6 at Settings of 10 Flat Pricing Functions



Continued on next page

Table B.25 – continued from previous page

No.	Query Plans
#5	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p>
#6	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p>
#7	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p>
#8	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p>
#9	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p>

Continued on next page

Table B.25 – continued from previous page

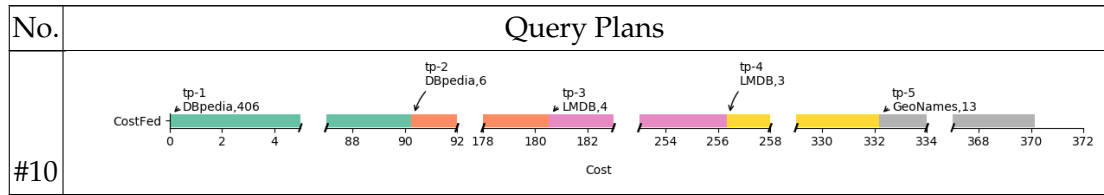
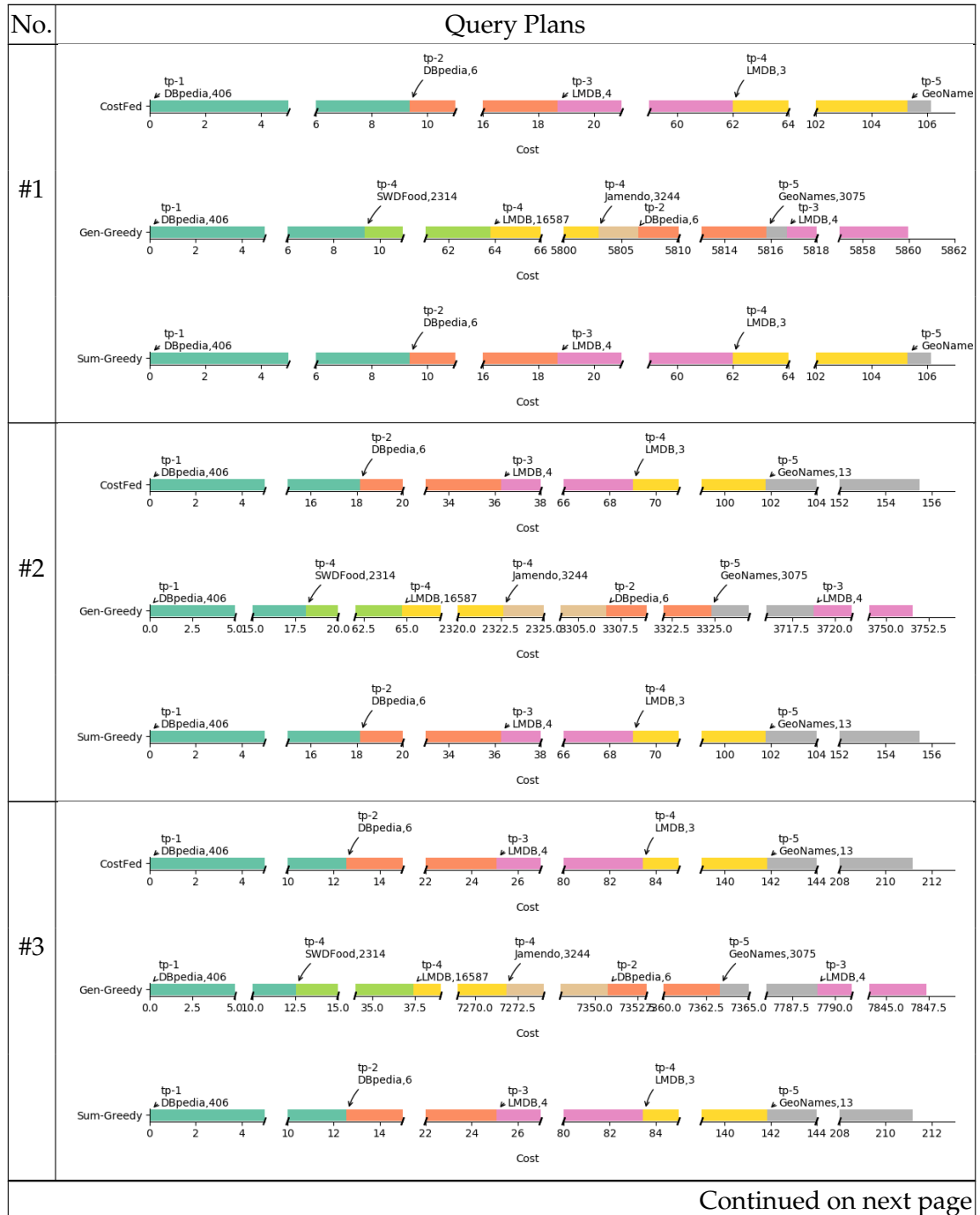


TABLE B.26: Query Plans of Query LD6 at Settings of 10 freemium Pricing Functions



Continued on next page

Table B.26 – continued from previous page

No.	Query Plans
#4	<p>CostFed: tp-1 DBpedia,406 (0-4), tp-2 DBpedia,6 (305.0-307.5), tp-3 LMB,4 (310.0-312.5), tp-4 LMB,3 (362-364), tp-5 GeoNames,13 (418-420), 422-494, 496-498</p> <p>Gen-Greedy: tp-1 DBpedia,406 (0-4), tp-4 SWDFood,2314 (304-306), tp-4 LMB,16587 (308-344), tp-4 Jamendo,3244 (346-348), tp-4 LMB,3244 (4070-4072), tp-2 DBpedia,6 (4074-4080), tp-5 GeoNames,3075 (4085-4160), tp-3 LMB,4 (4162-4164), 4214-4216, 4218</p> <p>Sum-Greedy: tp-1 DBpedia,406 (0-4), tp-2 DBpedia,6 (305.0-307.5), tp-3 LMB,4 (310.0-312.5), tp-4 LMB,3 (362-364), tp-5 GeoNames,13 (418-420), 422-494, 496-498</p>
#5	<p>CostFed: tp-1 DBpedia,406 (0-4), tp-2 DBpedia,6 (76-78), tp-3 LMB,4 (154-156), tp-4 LMB,3 (160-162), tp-5 GeoNames,13 (164-166), 168-192, 194-196</p> <p>Gen-Greedy: tp-1 DBpedia,406 (0.0-2.5), tp-4 SWDFood,2314 (5.075-77.5), tp-4 LMB,16587 (80.0-105.0), tp-4 Jamendo,3244 (107.5-4632.5), tp-2 DBpedia,6 (4635.0-4675.0), tp-5 GeoNames,3075 (4752.5-4755.0), tp-3 LMB,4 (4775.0-4780.0), 4785.0-4787.5</p> <p>Sum-Greedy: tp-1 DBpedia,406 (0-4), tp-2 DBpedia,6 (76-78), tp-3 LMB,4 (154-156), tp-4 LMB,3 (160-162), tp-5 GeoNames,13 (164-166), 168-192, 194-196</p>
#6	<p>CostFed: tp-1 DBpedia,406 (0-4), tp-2 DBpedia,6 (26-28), tp-3 LMB,4 (54-56), tp-4 LMB,3 (78-80), tp-5 GeoNames,13 (82-102), 104-124, 126-128</p> <p>Gen-Greedy: tp-1 DBpedia,406 (0.0-2.5), tp-2 DBpedia,6 (5.025-27.5), tp-4 SWDFood,2314 (30.0-55.0), tp-4 LMB,16587 (57.5-1000.0), tp-4 Jamendo,3244 (1002.5-1005.0), tp-5 GeoNames,3075 (1077.5-5130.0), tp-3 LMB,4 (5132.5-5135.0), 5137.5-6797.5, 6800.0-6820.0, 6822.5</p> <p>Sum-Greedy: tp-1 DBpedia,406 (0-4), tp-2 DBpedia,6 (26-28), tp-3 LMB,4 (54-56), tp-4 LMB,3 (78-80), tp-5 GeoNames,13 (82-102), 104-124, 126-128</p>
#7	<p>CostFed: tp-1 DBpedia,406 (0-4), tp-2 DBpedia,6 (38-40), tp-3 LMB,4 (80-82), tp-4 LMB,3 (120-122), tp-5 GeoNames,13 (124-160), 162-180, 182-184</p> <p>Gen-Greedy: tp-1 DBpedia,406 (0.0-2.5), tp-4 SWDFood,2314 (40.0-42.5), tp-4 LMB,16587 (525.0-527.5), tp-4 Jamendo,3244 (9302.5-9305.0), tp-2 DBpedia,6 (9380.0-9382.5), tp-5 GeoNames,3075 (9420.0-9422.5), tp-3 LMB,4 (9440.0-9442.5), 9480.0-9482.5</p> <p>Sum-Greedy: tp-1 DBpedia,406 (0-4), tp-2 DBpedia,6 (38-40), tp-3 LMB,4 (80-82), tp-4 LMB,3 (120-122), tp-5 GeoNames,13 (124-160), 162-180, 182-184</p>

Continued on next page

Table B.26 – continued from previous page

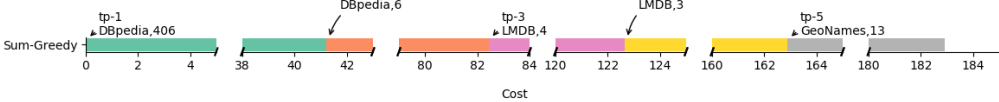
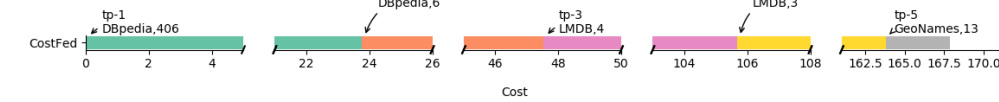

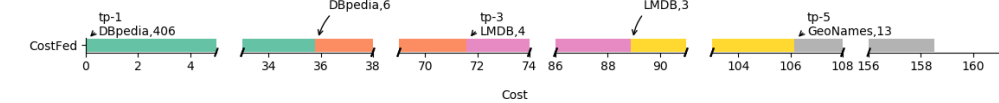
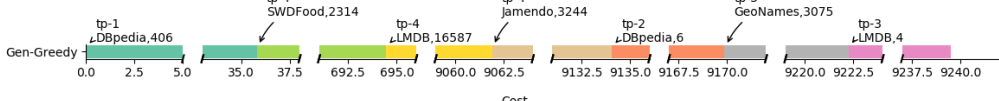
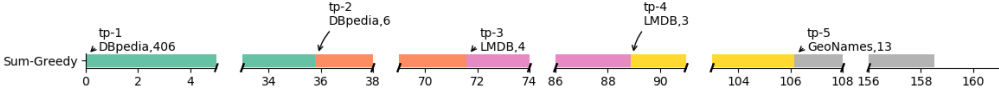
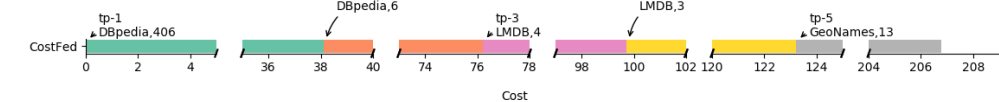
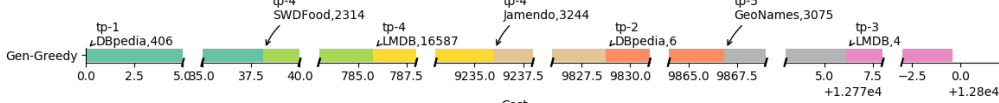
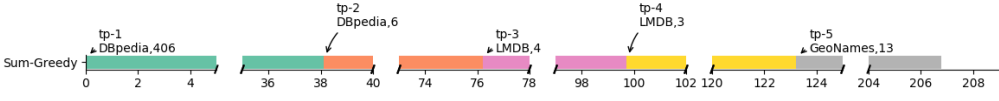
No.	Query Plans
	 <p>Sum-Greedy</p> <p>Cost: 0 2 4 38 40 42 80 82 84 120 122 124 160 162 164 180 182 184</p>
#8	 <p>CostFed</p> <p>Cost: 0 2 4 22 24 26 46 48 50 104 106 108 162.5 165.0 167.5 170.0</p>  <p>Gen-Greedy</p> <p>Cost: 0 2 4 22 24 26 46 48 50 128 130 132 5862 5864 5866 5940 5945 5998 6000 6002</p>  <p>Sum-Greedy</p> <p>Cost: 0 2 4 22 24 26 46 48 50 104 106 108 162.5 165.0 167.5 170.0</p>
#9	 <p>CostFed</p> <p>Cost: 0 2 4 34 36 38 70 72 74 86 88 90 104 106 108 156 158 160</p>  <p>Gen-Greedy</p> <p>Cost: 0.0 2.5 5.0 35.0 37.5 692.5 695.0 9060.0 9062.5 9132.5 9135.0 9167.5 9170.0 9220.0 9222.5 9237.5 9240.0</p>  <p>Sum-Greedy</p> <p>Cost: 0 2 4 34 36 38 70 72 74 86 88 90 104 106 108 156 158 160</p>
#10	 <p>CostFed</p> <p>Cost: 0 2 4 36 38 40 74 76 78 98 100 102 120 122 124 204 206 208</p>  <p>Gen-Greedy</p> <p>Cost: 0.0 2.5 5.085 37.5 40.0 785.0 787.5 9235.0 9237.5 9827.5 9830.0 9865.0 9867.5 5.0 7.5 -2.5 0.0 +1.277e4 +1.28e4</p>  <p>Sum-Greedy</p> <p>Cost: 0 2 4 36 38 40 74 76 78 98 100 102 120 122 124 204 206 208</p>

TABLE B.27: Query Plans of Query LD6 at Settings of 10 per Pricing Functions

No.	Query Plans
#1	<p>CostFed: tp-1 DBpedia,406 (0-5), tp-2 DBpedia,6 (50-52), tp-3 LMDB,4 (52-54), tp-4 LMDB,3 (54-58), tp-5 GeoNames,13 (58-60), (66-69)</p> <p>Gen-Greedy: tp-1 DBpedia,406 (0-5), tp-2 DBpedia,6 (50-52), tp-3 SWDFood,2314 (52-54), tp-4 LMDB,16587 (54-56), tp-4 Jamendo,3244 (56-58), tp-5 GeoNames,3075 (58-60), tp-3 LMDB,4 (60-62), (18735-18740)</p> <p>Sum-Greedy: tp-1 DBpedia,406 (0-5), tp-2 DBpedia,6 (50-52), tp-3 LMDB,4 (52-54), tp-4 LMDB,3 (54-58), tp-5 GeoNames,13 (58-60), (66-69)</p>
#2	<p>CostFed: tp-1 DBpedia,406 (0-5), tp-2 DBpedia,6 (240-242), tp-3 LMDB,4 (242-244), tp-4 LMDB,3 (244-248), tp-5 GeoNames,13 (248-250), (252-254)</p> <p>Sum-Greedy: tp-1 DBpedia,406 (0-5), tp-2 DBpedia,6 (240-242), tp-3 LMDB,4 (242-244), tp-4 LMDB,3 (244-248), tp-5 GeoNames,13 (248-250), (252-254)</p>
#3	<p>CostFed: tp-1 DBpedia,406 (0-5), tp-2 DBpedia,6 (212.5-215.0), tp-3 LMDB,4 (215.0-217.5), tp-4 LMDB,3 (217.5-220.0), tp-5 GeoNames,13 (220.0-222.5), (225.0-232), (233-234), (235-237)</p> <p>Gen-Greedy: tp-1 DBpedia,406 (0-5), tp-4 SWDFood,2314 (212-214), tp-4 LMDB,16587 (216-218), tp-4 Jamendo,3244 (218-220), tp-5 GeoNames,3075 (220-222), tp-2 DBpedia,6 (222-224), tp-3 LMDB,4 (224-226), (18175-18180), (0-5), (+2.09e4)</p> <p>Sum-Greedy: tp-1 DBpedia,406 (0-5), tp-2 DBpedia,6 (212.5-215.0), tp-3 LMDB,4 (215.0-217.5), tp-4 LMDB,3 (217.5-220.0), tp-5 GeoNames,13 (220.0-222.5), (225.0-232), (233-234), (235-237)</p>
#4	<p>CostFed: tp-1 DBpedia,406 (0-5), tp-2 DBpedia,6 (353-354), tp-3 LMDB,4 (354-356), tp-4 LMDB,3 (356-358), tp-5 GeoNames,13 (358-360), (362-364), (366-368)</p> <p>Sum-Greedy: tp-1 DBpedia,406 (0-5), tp-2 DBpedia,6 (353-354), tp-3 LMDB,4 (354-356), tp-4 LMDB,3 (356-358), tp-5 GeoNames,13 (358-360), (362-364), (366-368)</p>
#5	<p>CostFed: tp-1 DBpedia,406 (0-5), tp-2 DBpedia,6 (396-397), tp-4 LMDB,3 (397-398), tp-3 LMDB,4 (398-399), tp-5 GeoNames,13 (399-401), (402-403), (404-405), (406-407), (408-409), (410-412)</p>
#5	Continued on next page

Table B.27 – continued from previous page

Query Plans

The figure illustrates query plans for three queries (tp-1, tp-2, tp-3) across three different cost models: Gen-Greedy, Sum-Greedy, and CostFed. The plans are visualized as horizontal bars with segments representing different database tables and their associated costs. The x-axis represents the total cost, and the y-axis represents the query number. The Gen-Greedy and Sum-Greedy models show a wide range of costs, while the CostFed model shows a much narrower range, indicating more consistent and lower costs.

Query 1 (tp-1): DBpedia, 406

Query 2 (tp-2): DBpedia, 6

Query 3 (tp-3): LMDb, 4

Query 4 (tp-4): SWDFood, 2314; LMDb, 16587; Jamendo, 3244

Query 5 (tp-5): GeoNames, 3075

Cost Models:

- Gen-Greedy:** Shows a wide range of costs, with the total cost for Query 1 being 406, for Query 2 being 6, and for Query 3 being 4. The total cost for Query 4 is 16587, and for Query 5 is 3075.
- Sum-Greedy:** Shows a similar range of costs, with the total cost for Query 1 being 406, for Query 2 being 6, and for Query 3 being 4. The total cost for Query 4 is 16587, and for Query 5 is 3075.
- CostFed:** Shows a much narrower range of costs, with the total cost for Query 1 being 406, for Query 2 being 6, and for Query 3 being 4. The total cost for Query 4 is 16587, and for Query 5 is 3075.

Table B.27 – continued from previous page

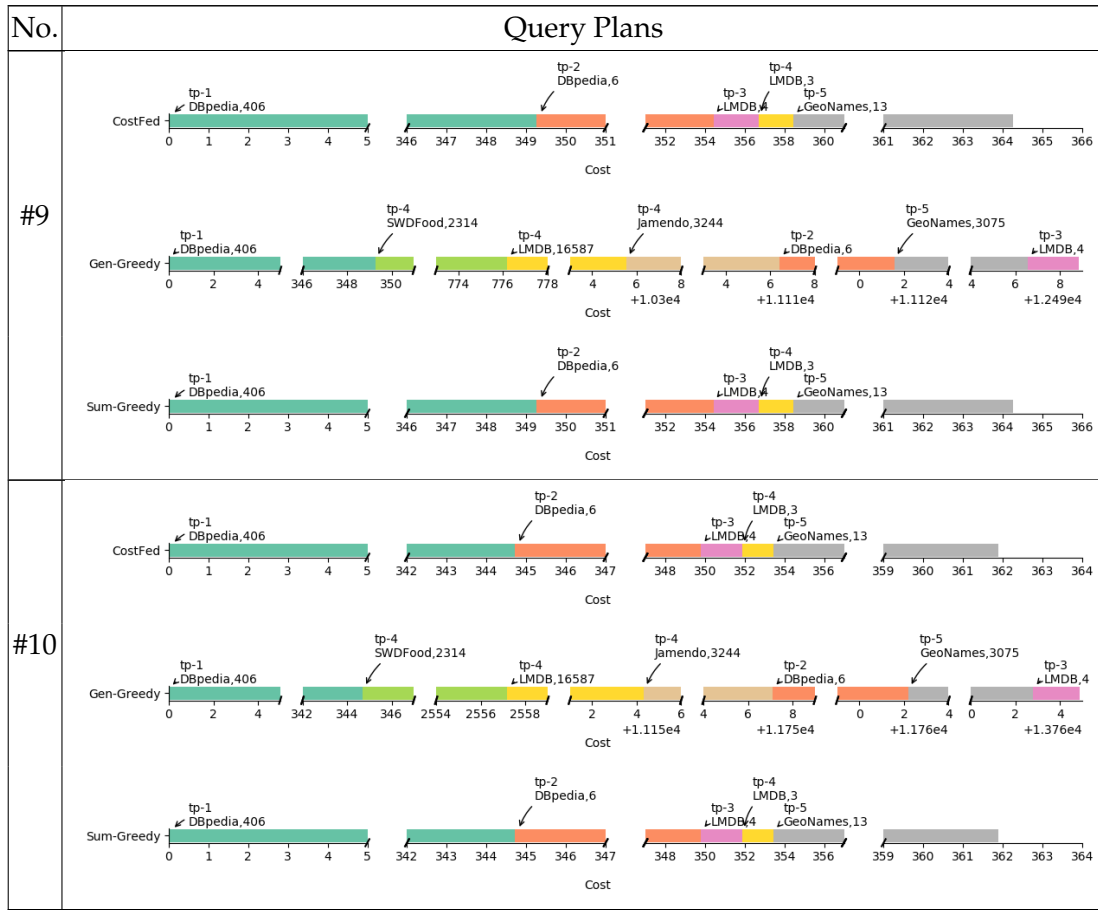
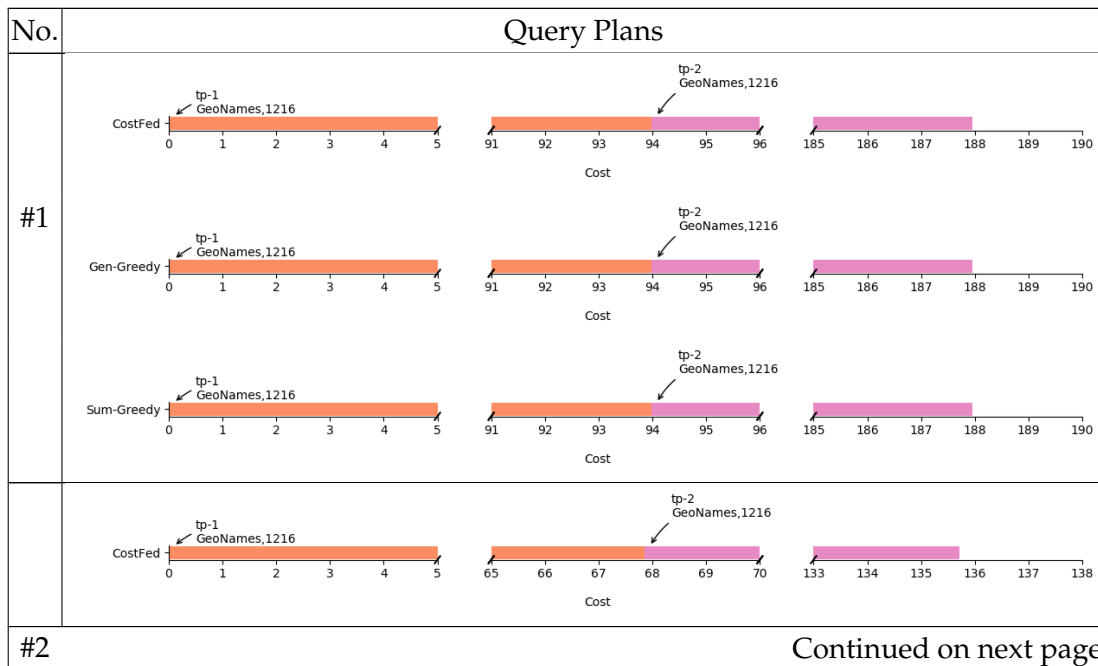


TABLE B.28: Query Plans of Query LD7 at Settings of 10 Flat Pricing Functions



Continued on next page

Table B.28 – continued from previous page

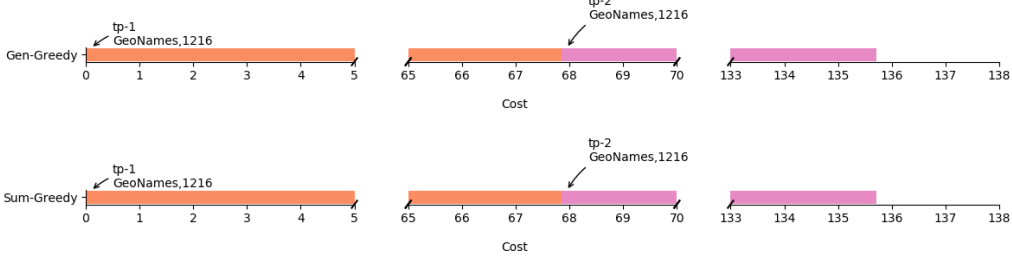
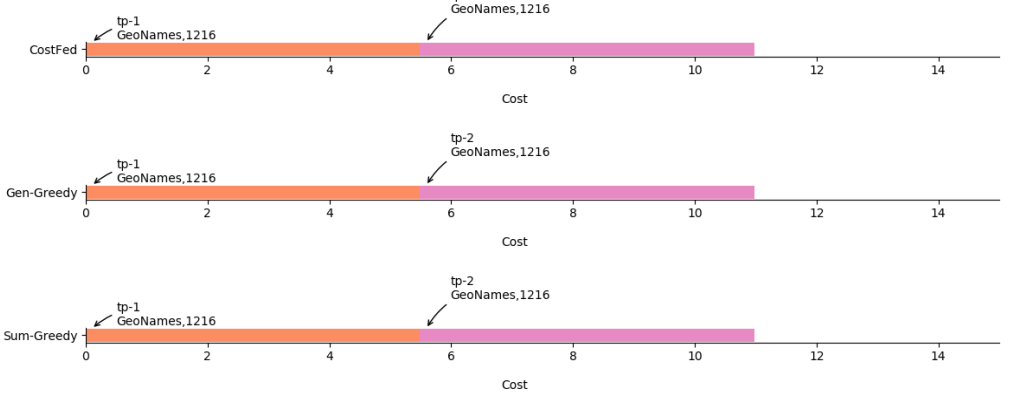
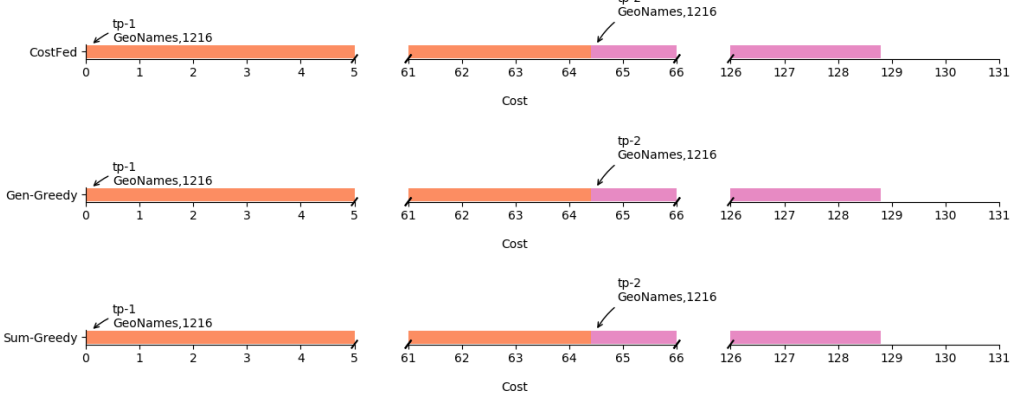
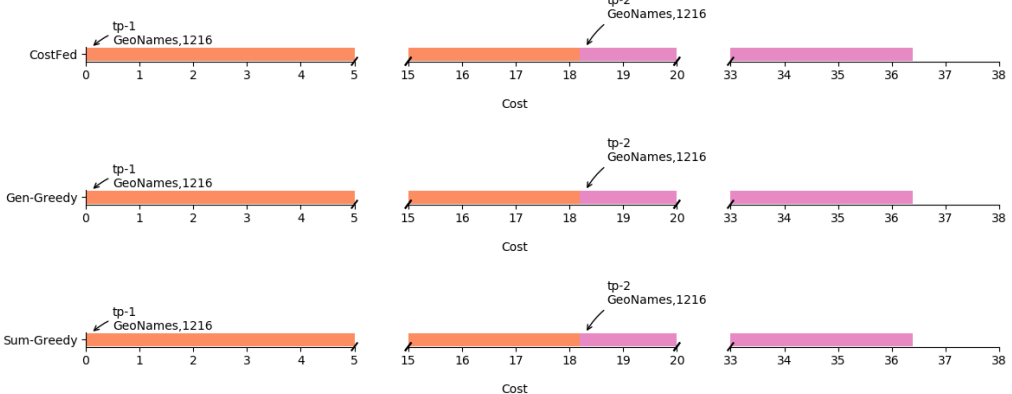
No.	Query Plans
	 <p>Gen-Greedy: tp-1 GeoNames,1216 (0-5), tp-2 GeoNames,1216 (65-70), Cost (133-138)</p> <p>Sum-Greedy: tp-1 GeoNames,1216 (0-5), tp-2 GeoNames,1216 (65-70), Cost (133-138)</p>
#3	 <p>CostFed: tp-1 GeoNames,1216 (0-5), tp-2 GeoNames,1216 (6-11), Cost (0-14)</p> <p>Gen-Greedy: tp-1 GeoNames,1216 (0-5), tp-2 GeoNames,1216 (6-11), Cost (0-14)</p> <p>Sum-Greedy: tp-1 GeoNames,1216 (0-5), tp-2 GeoNames,1216 (6-11), Cost (0-14)</p>
#4	 <p>CostFed: tp-1 GeoNames,1216 (0-5), tp-2 GeoNames,1216 (61-66), Cost (126-131)</p> <p>Gen-Greedy: tp-1 GeoNames,1216 (0-5), tp-2 GeoNames,1216 (61-66), Cost (126-131)</p> <p>Sum-Greedy: tp-1 GeoNames,1216 (0-5), tp-2 GeoNames,1216 (61-66), Cost (126-131)</p>
#5	 <p>CostFed: tp-1 GeoNames,1216 (0-5), tp-2 GeoNames,1216 (15-20), Cost (33-38)</p> <p>Gen-Greedy: tp-1 GeoNames,1216 (0-5), tp-2 GeoNames,1216 (15-20), Cost (33-38)</p> <p>Sum-Greedy: tp-1 GeoNames,1216 (0-5), tp-2 GeoNames,1216 (15-20), Cost (33-38)</p>
Continued on next page	

Table B.28 – continued from previous page

No.	Query Plans
#6	
#7	
#8	
#9	

Continued on next page

Table B.28 – continued from previous page

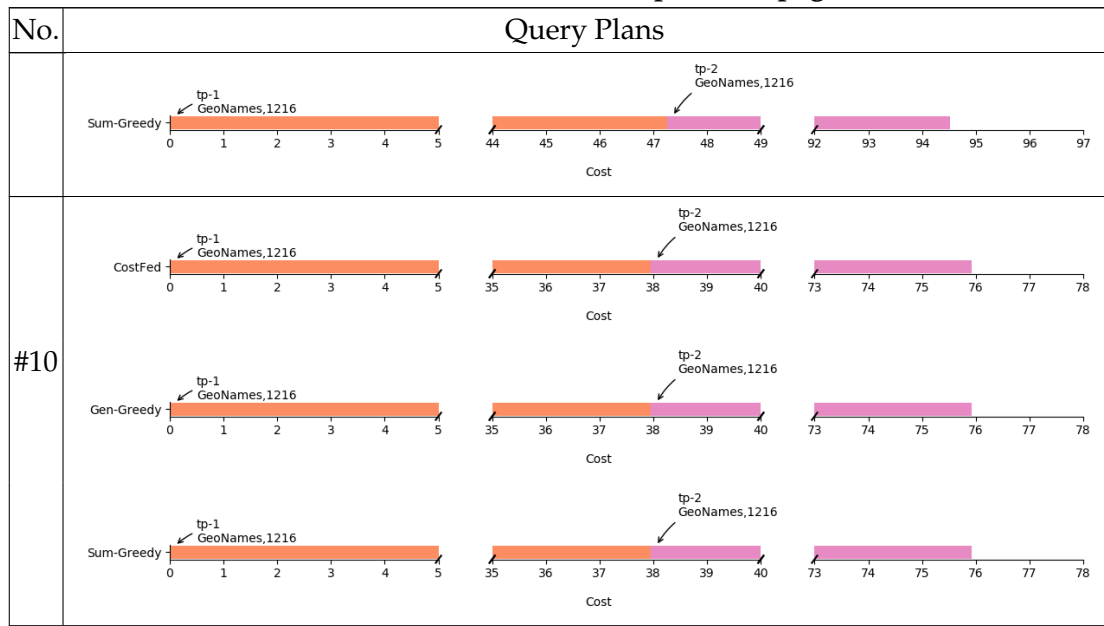
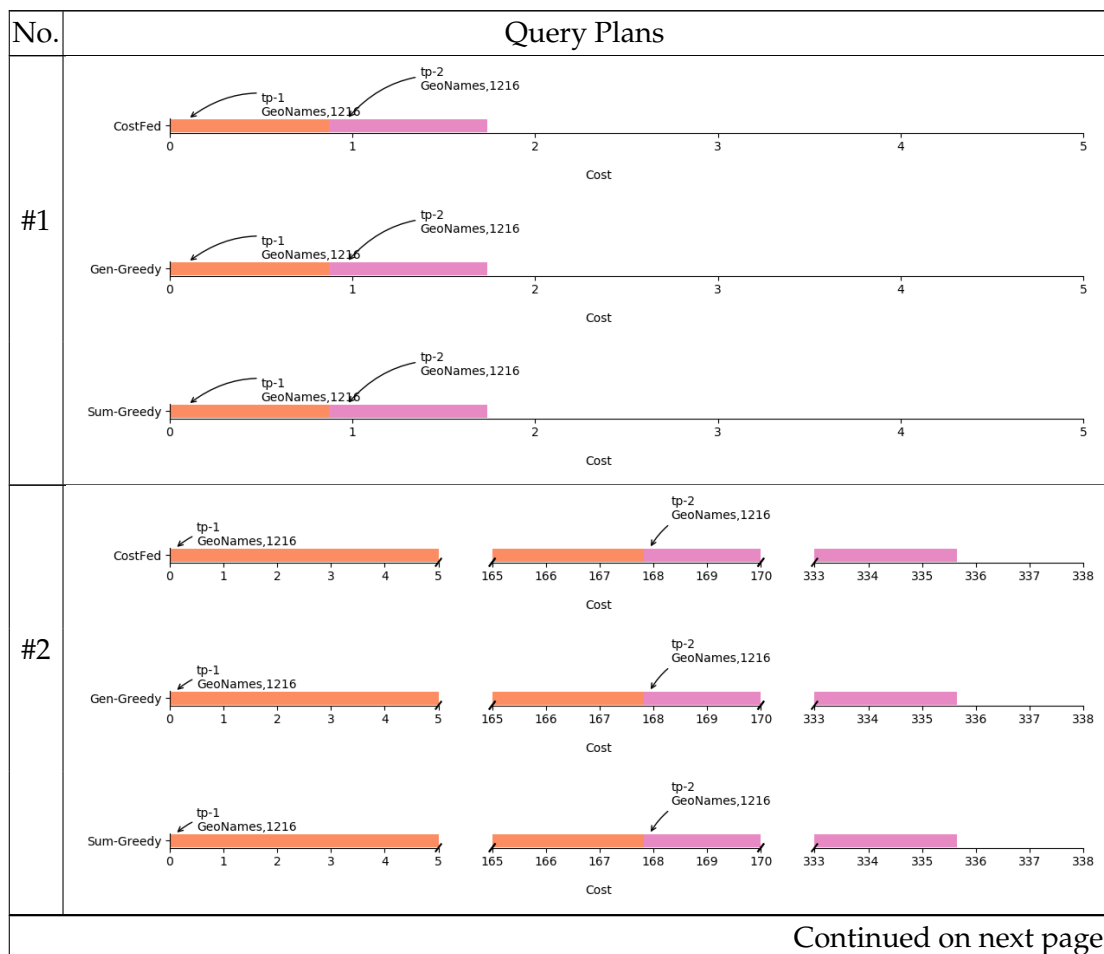


TABLE B.29: Query Plans of Query LD7 at Settings of 10 freemium Pricing Functions



Continued on next page

Table B.29 – continued from previous page

No.	Query Plans
#3	
#4	
#5	
#6	

Continued on next page

Table B.29 – continued from previous page

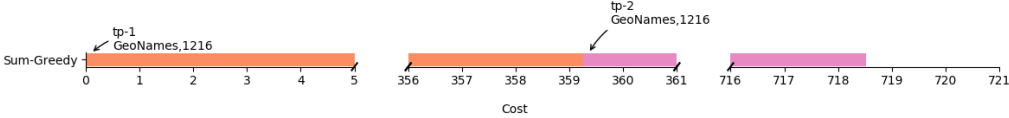
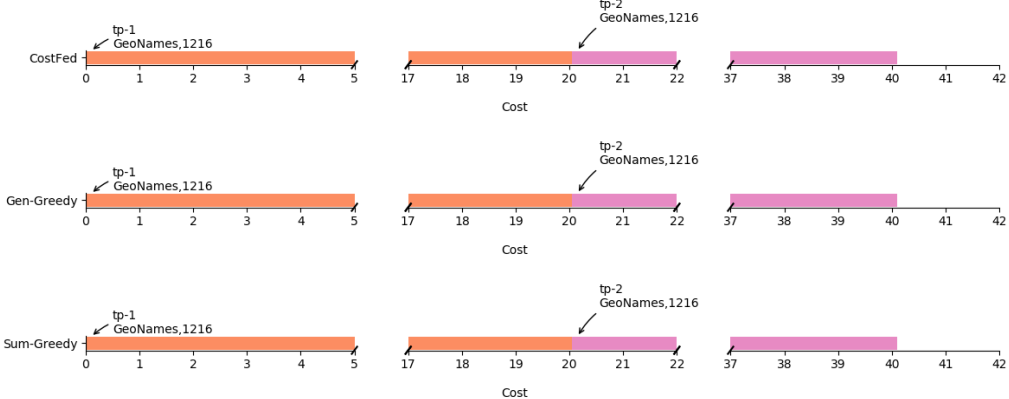
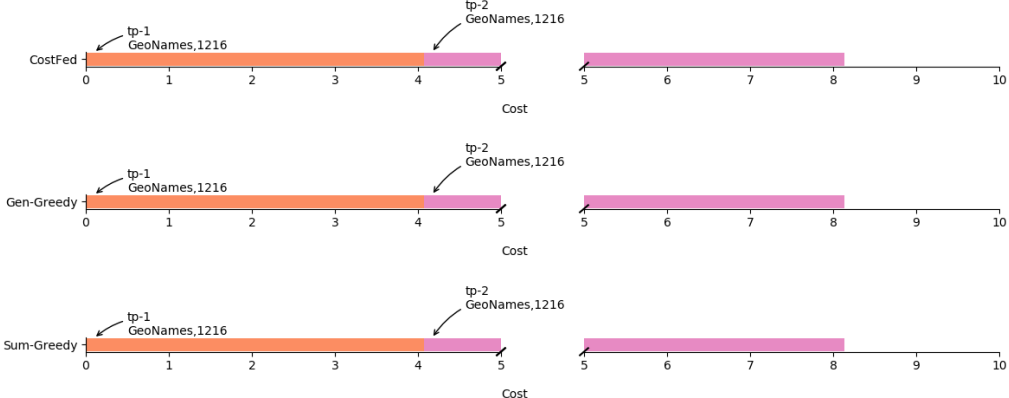
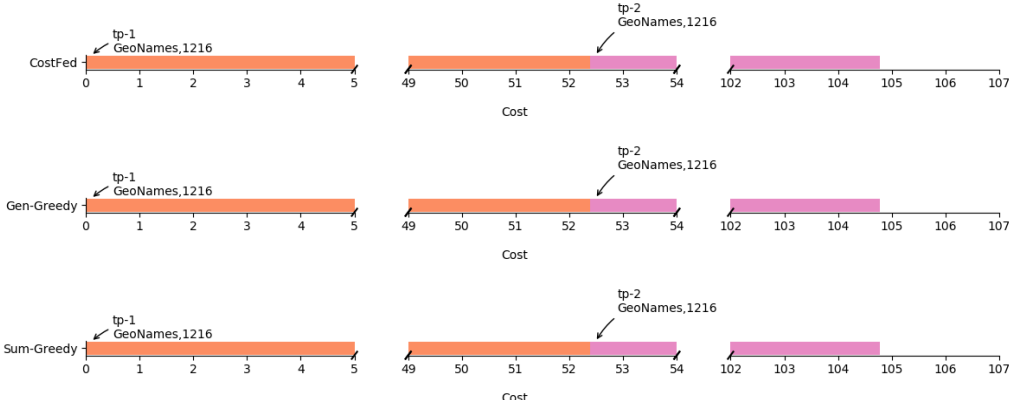
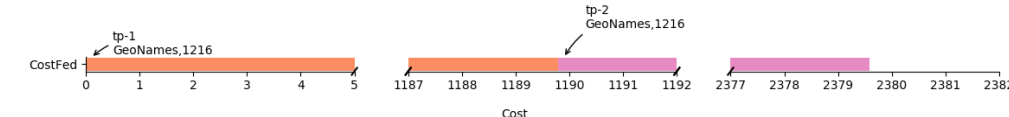
No.	Query Plans
	 <p>Sum-Greedy</p> <p>tp-1 GeoNames,1216</p> <p>tp-2 GeoNames,1216</p> <p>Cost</p>
#7	 <p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>tp-1 GeoNames,1216</p> <p>tp-2 GeoNames,1216</p> <p>Cost</p>
#8	 <p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>tp-1 GeoNames,1216</p> <p>tp-2 GeoNames,1216</p> <p>Cost</p>
#9	 <p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>tp-1 GeoNames,1216</p> <p>tp-2 GeoNames,1216</p> <p>Cost</p>
	 <p>CostFed</p> <p>tp-1 GeoNames,1216</p> <p>tp-2 GeoNames,1216</p> <p>Cost</p>
#10	Continued on next page

Table B.29 – continued from previous page

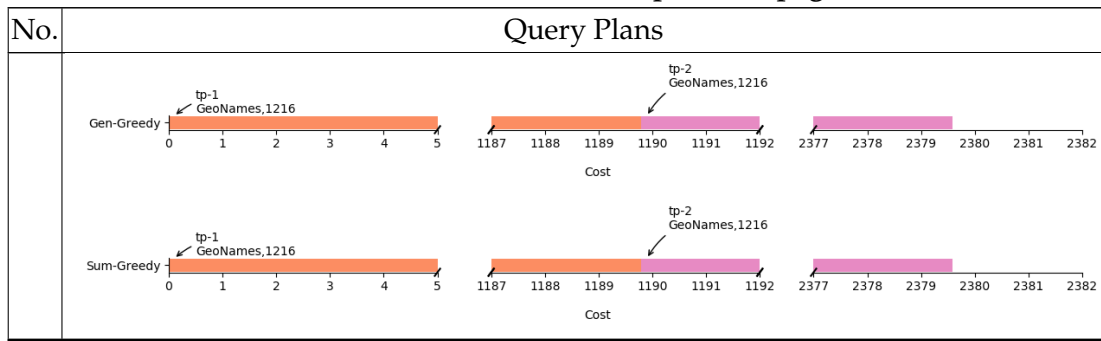
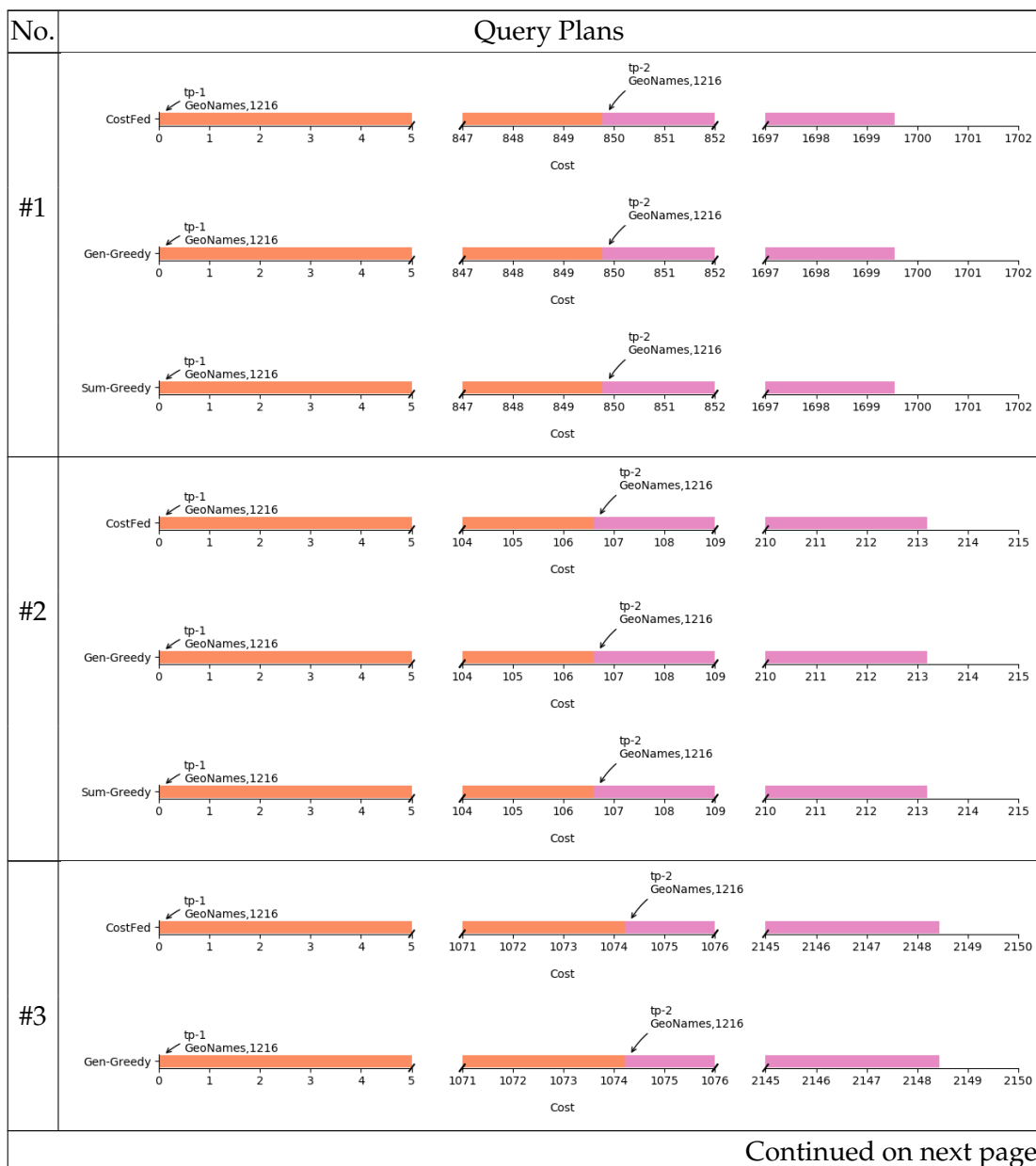


TABLE B.30: Query Plans of Query LD7 at Settings of 10 per Pricing Functions



Continued on next page

Table B.30 – continued from previous page

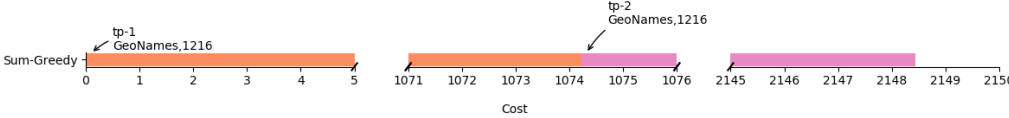
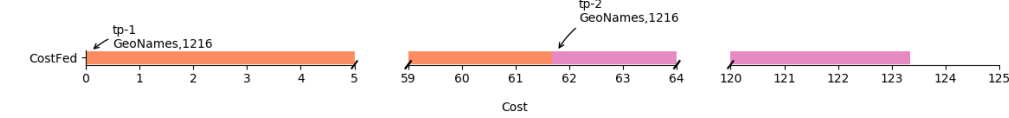
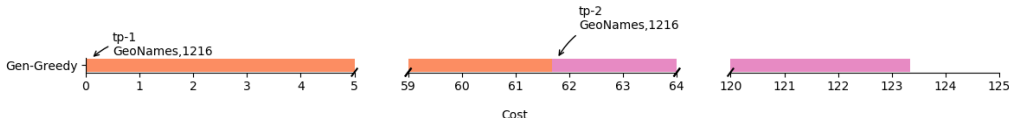
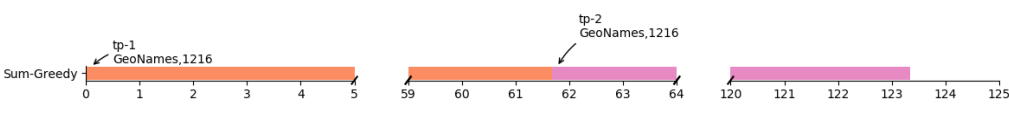
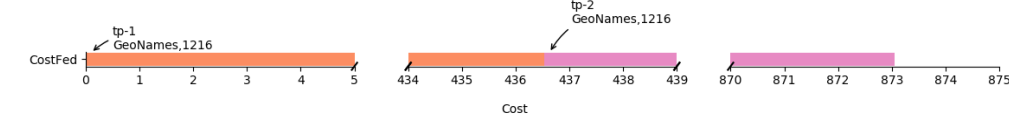
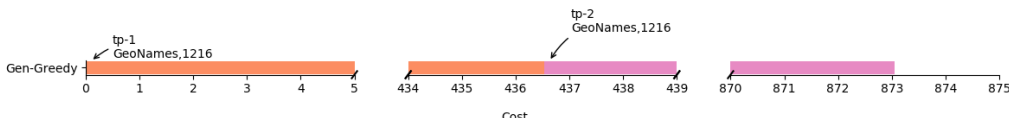
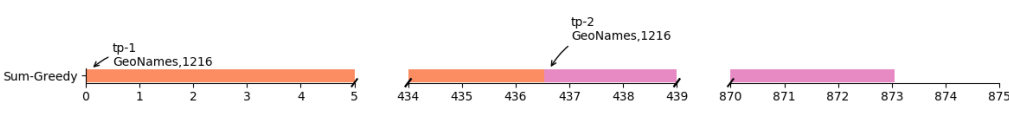
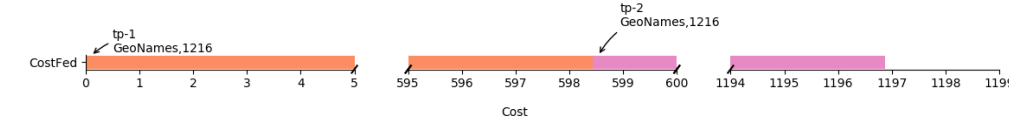
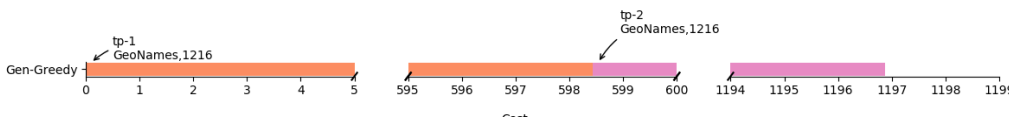
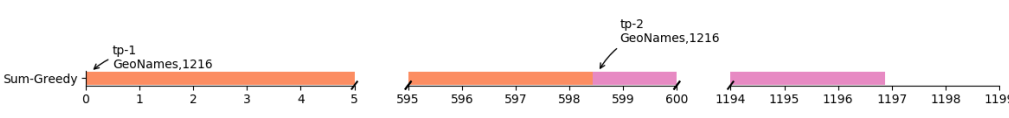
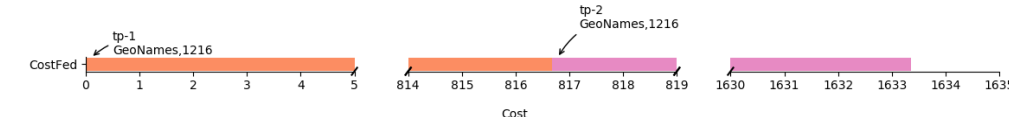
No.	Query Plans
	
#4	  
#5	  
#6	  
	
#7	Continued on next page

Table B.30 – continued from previous page

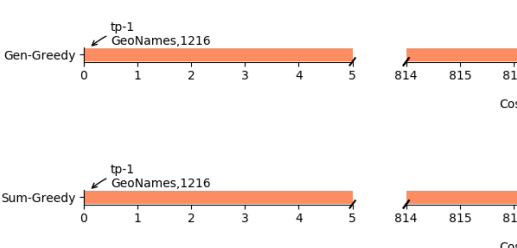
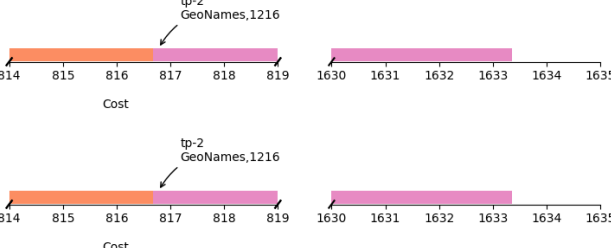
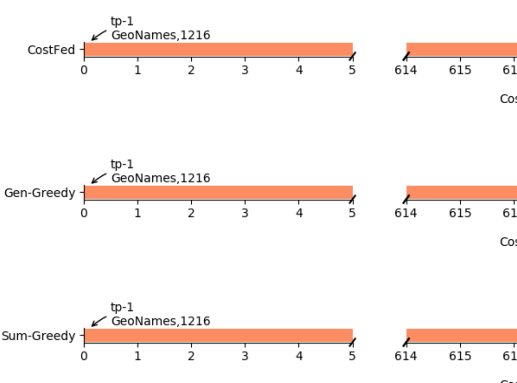
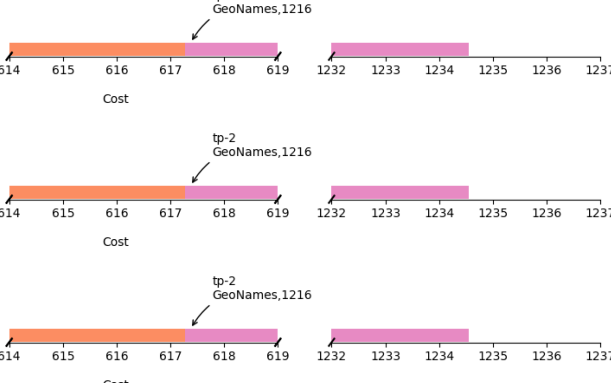
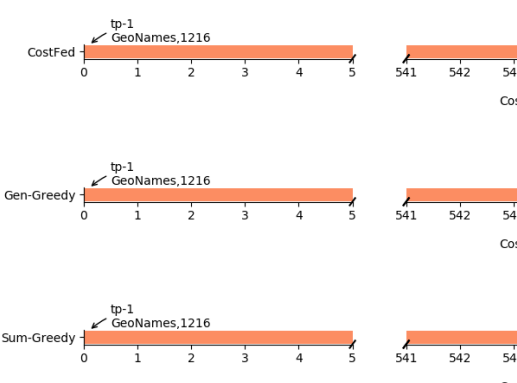

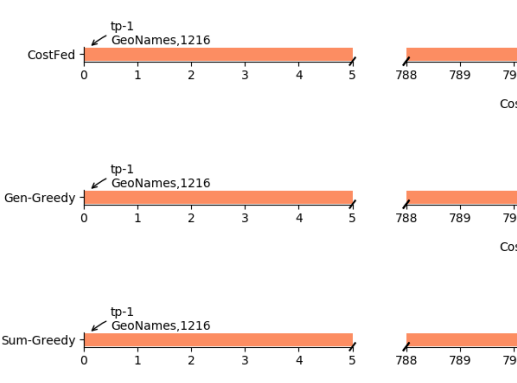
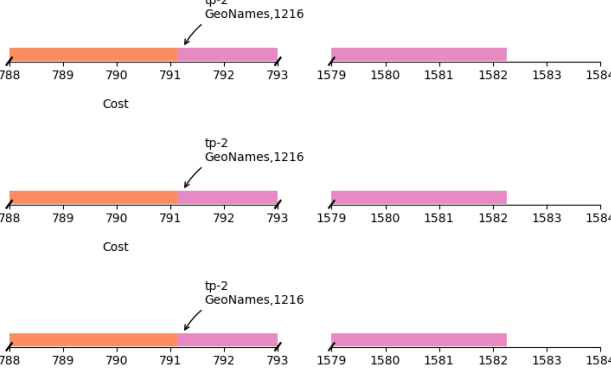
No.	Query Plans
	 
#8	 
#9	 
#10	 

TABLE B.31: Query Plans of Query LD8 at Settings of 10 Flat Pricing Functions

No.	Query Plans
#1	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p>
#2	<p>CostFed</p>
#3	<p>CostFed</p>
#4	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p>
#5	<p>CostFed</p>
#6	<p>CostFed</p> <p>Gen-Greedy</p>

Continued on next page

Table B.31 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#8	<p>CostFed</p> <p>Cost</p>
#9	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#10	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>

TABLE B.32: Query Plans of Query LD8 at Settings of 10 freemium Pricing Functions

No.	Query Plans
#1	<p>CostFed: 0 2 4 72 74 76 148 150 152 222 224 226 232 234 236 240 242 244</p> <p>Gen-Greedy: 0 2 4 72 74 76 148 150 152 222 224 226 232 234 236 240 242 244</p> <p>Sum-Greedy: 0 2 4 72 74 76 148 150 152 222 224 226 232 234 236 240 242 244</p>
#2	<p>CostFed: 0 2 4 88 90 92 180 182 184 270 272 274 288 290 292 306 308 310</p> <p>Gen-Greedy: 0 2 4 88 90 92 180 182 184 270 272 274 288 290 292 306 308 310</p> <p>Sum-Greedy: 0 2 4 88 90 92 180 182 184 270 272 274 288 290 292 306 308 310</p>
#3	<p>CostFed: 0 2 4 34 36 38 70 72 74 108 110 112 120 122 124 132 134 136</p> <p>Gen-Greedy: 0 2 4 34 36 38 70 72 74 108 110 112 120 122 124 132 134 136</p> <p>Sum-Greedy: 0 2 4 34 36 38 70 72 74 108 110 112 120 122 124 132 134 136</p>
#4	<p>CostFed: 0 1 2 3 4 5 46 47 48 49 50 51 96 97 98 99 100 101 145 150 155 160</p> <p>Gen-Greedy: 0 1 2 3 4 5 46 47 48 49 50 51 96 97 98 99 100 101 145 150 155 160</p>

Continued on next page

Table B.32 – continued from previous page

No.	Query Plans
#5	
#6	
#7	
#8	Continued on next page

Table B.32 – continued from previous page

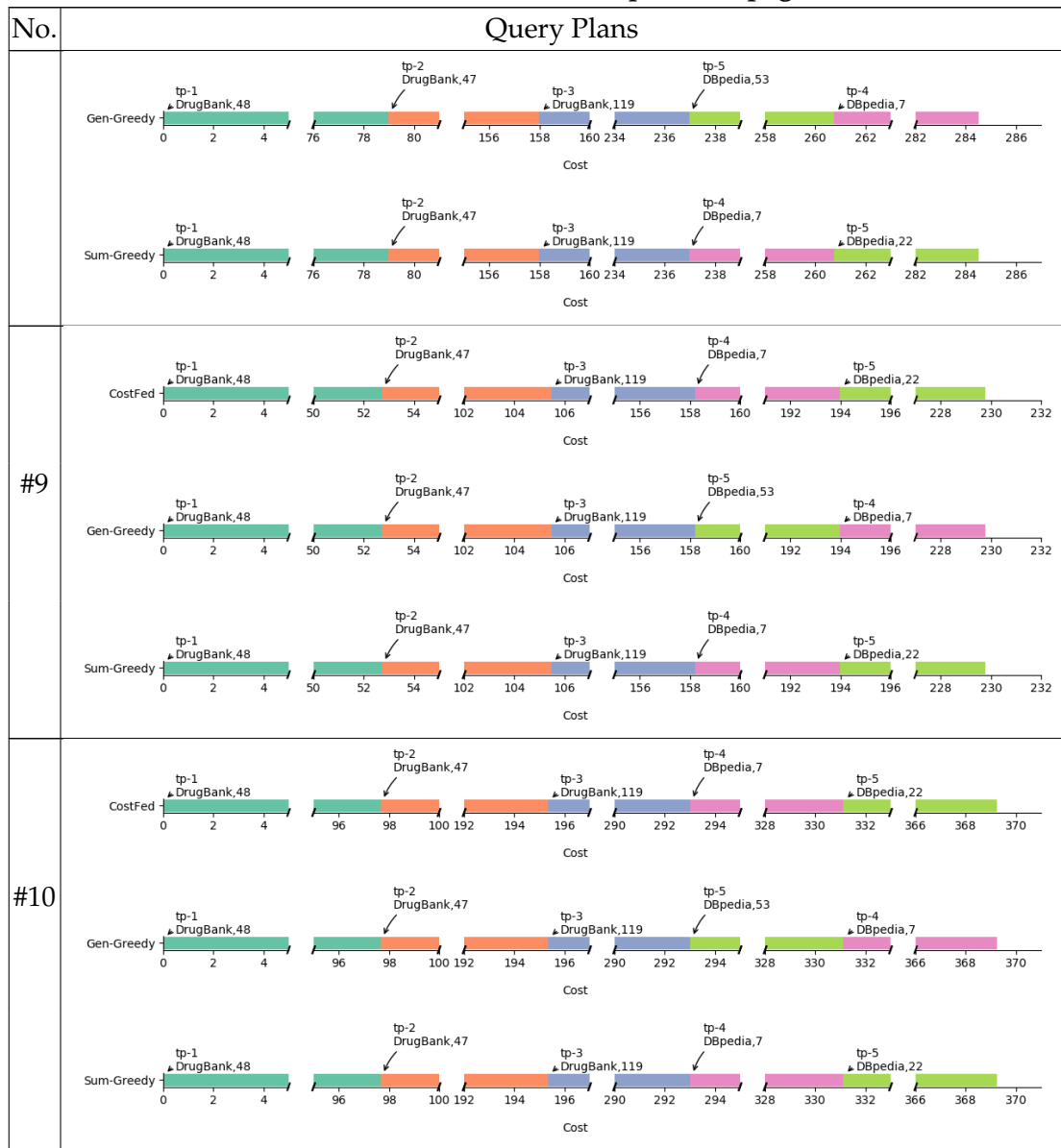
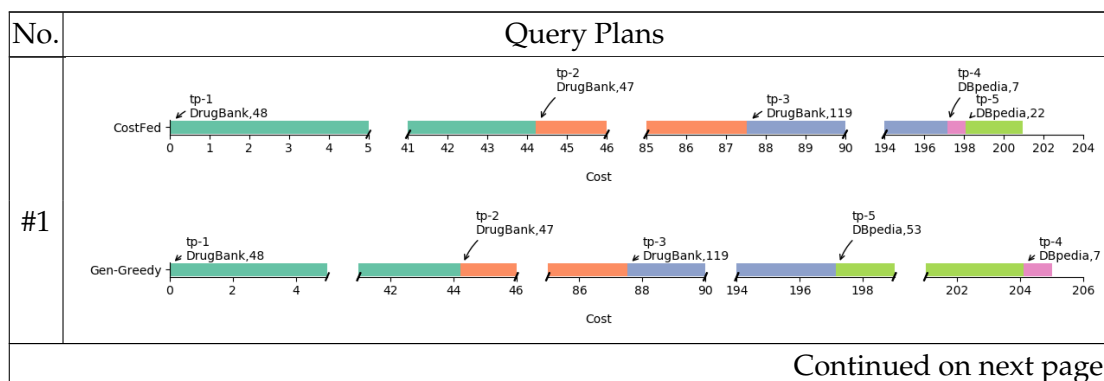


TABLE B.33: Query Plans of Query LD8 at Settings of 10 per Pricing Functions



Continued on next page

Table B.33 – continued from previous page

No.	Query Plans
#2	
#3	
#4	
#5	Continued on next page

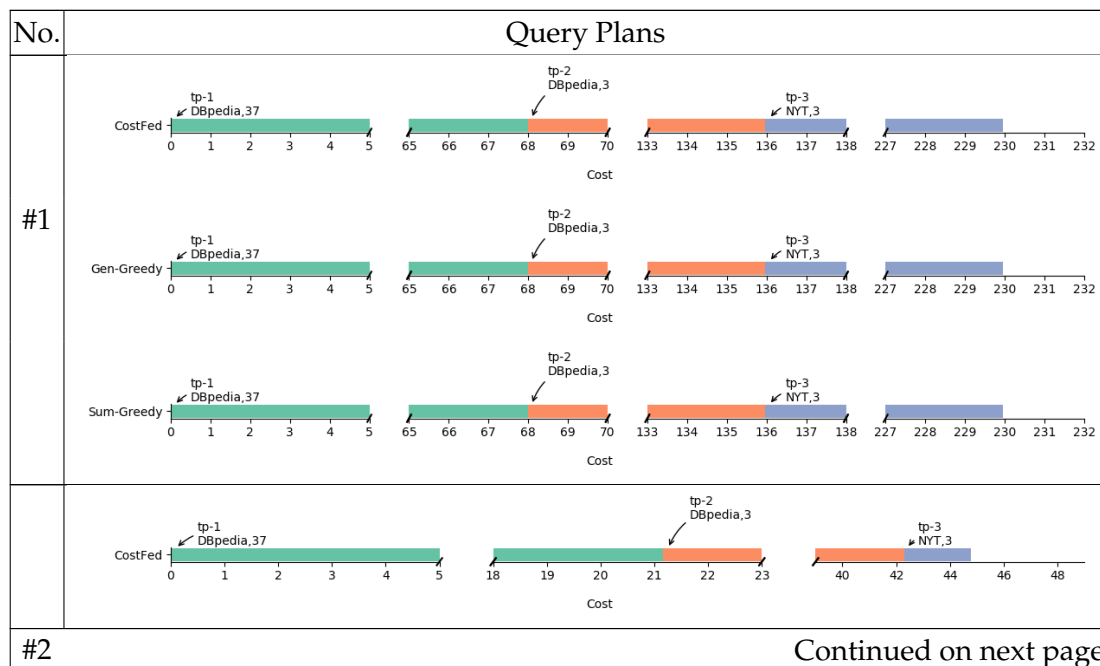
Table B.33 – continued from previous page

No.	Query Plans
#6	
#7	
#8	
Continued on next page	

Table B.33 – continued from previous page



TABLE B.34: Query Plans of Query LD10 at Settings of 10 Flat Pricing Functions



Continued on next page

Table B.34 – continued from previous page

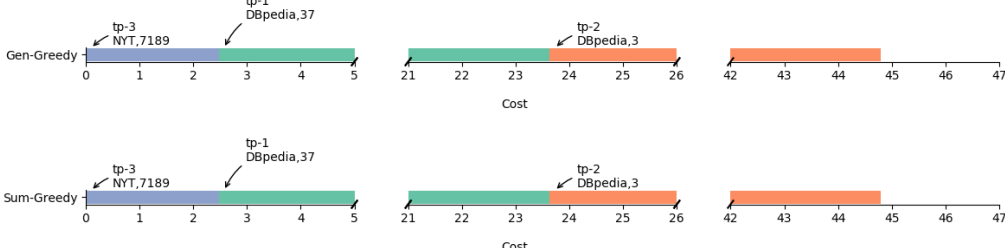
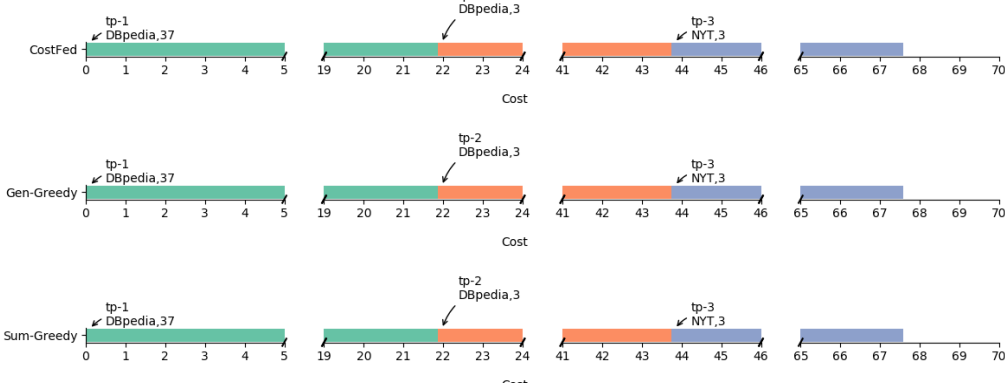
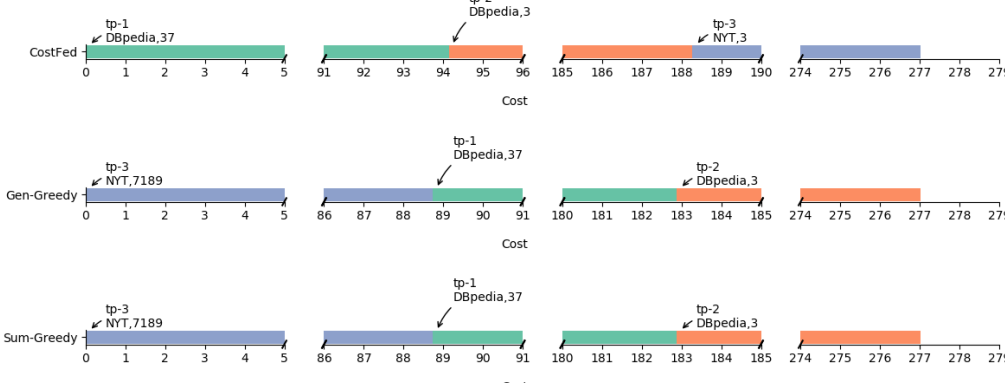
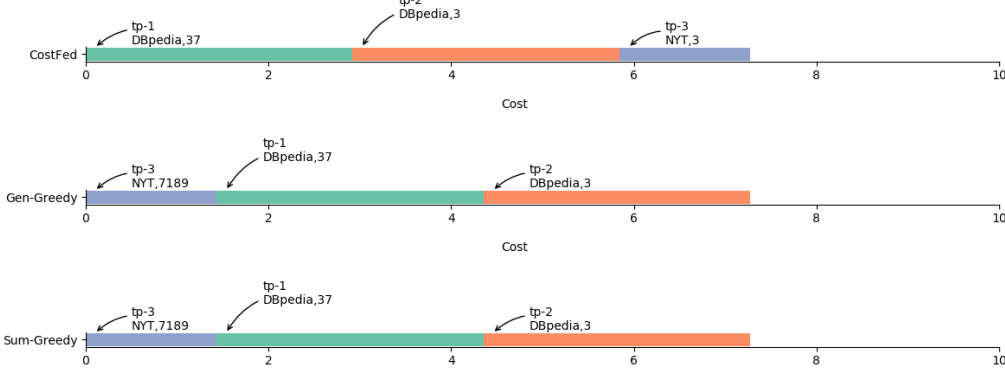
No.	Query Plans
	
#3	
#4	
#5	
Continued on next page	

Table B.34 – continued from previous page

No.	Query Plans
#6	<p>CostFed</p> <p>Cost</p>
	<p>Gen-Greedy</p> <p>Cost</p>
	<p>Sum-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Cost</p>
	<p>Gen-Greedy</p> <p>Cost</p>
	<p>Sum-Greedy</p> <p>Cost</p>
#8	<p>CostFed</p> <p>Cost</p>
	<p>Gen-Greedy</p> <p>Cost</p>
	<p>Sum-Greedy</p> <p>Cost</p>
#9	<p>CostFed</p> <p>Cost</p>
	<p>Gen-Greedy</p> <p>Cost</p>

Continued on next page

Table B.34 – continued from previous page

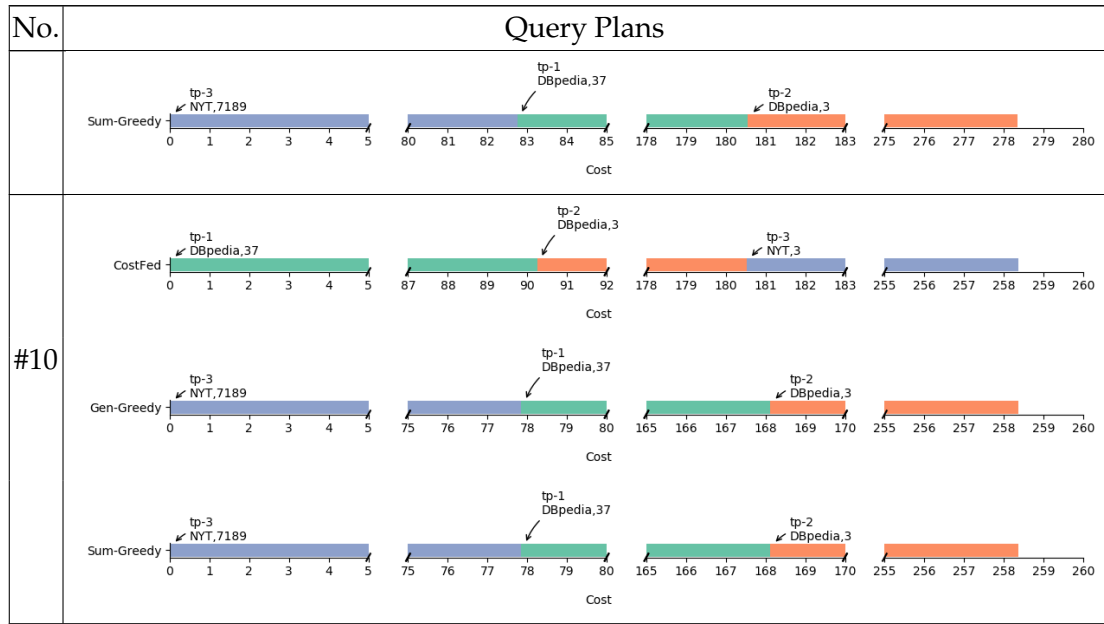
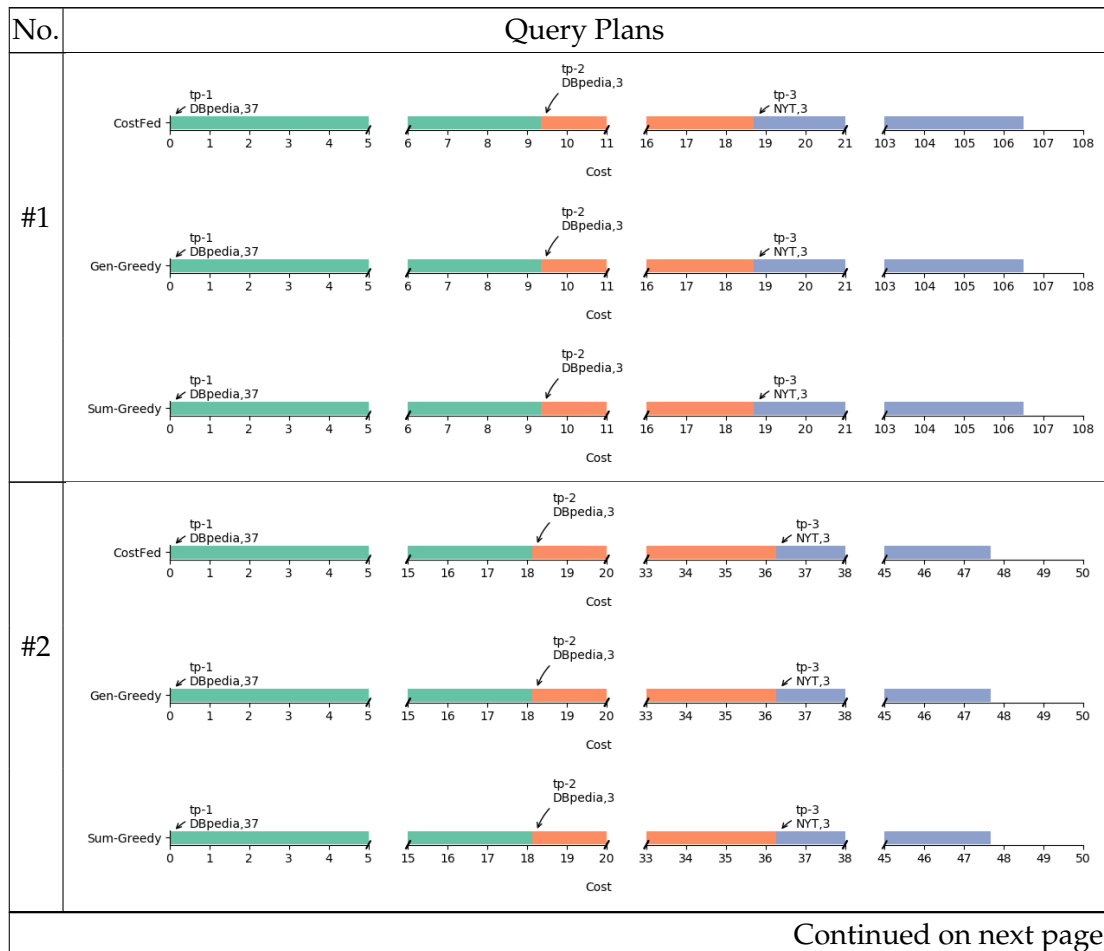


TABLE B.35: Query Plans of Query LD10 at Settings of 10 freemium Pricing Functions



Continued on next page

Table B.35 – continued from previous page

No.	Query Plans
#3	<p>CostFed: tp-1 DBpedia,37 (0-5), tp-2 DBpedia,3 (10-15), tp-3 NYT,3 (22-27), 104-109</p> <p>Gen-Greedy: tp-1 DBpedia,37 (0-5), tp-2 DBpedia,3 (10-15), tp-3 NYT,3 (22-27), 104-109</p> <p>Sum-Greedy: tp-1 DBpedia,37 (0-5), tp-2 DBpedia,3 (10-15), tp-3 NYT,3 (22-27), 104-109</p>
#4	<p>CostFed: tp-1 DBpedia,37 (0-5), tp-2 DBpedia,3 (5-10), tp-3 NYT,3 (92-97)</p> <p>Gen-Greedy: tp-1 DBpedia,37 (0-5), tp-3 NYT,7189 (5-202), tp-2 DBpedia,3 (202-208)</p> <p>Sum-Greedy: tp-1 DBpedia,37 (0-5), tp-2 DBpedia,3 (5-10), tp-3 NYT,3 (92-97)</p>
#5	<p>CostFed: tp-1 DBpedia,37 (0-5), tp-2 DBpedia,3 (75-80), tp-3 NYT,3 (153-158), 234-239</p> <p>Gen-Greedy: tp-1 DBpedia,37 (0-5), tp-2 DBpedia,3 (75-80), tp-3 NYT,3 (153-158), 234-239</p> <p>Sum-Greedy: tp-1 DBpedia,37 (0-5), tp-2 DBpedia,3 (75-80), tp-3 NYT,3 (153-158), 234-239</p>
#6	<p>CostFed: tp-1 DBpedia,37 (0-5), tp-2 DBpedia,3 (25-30), tp-3 NYT,3 (54-59), 119-124</p> <p>Gen-Greedy: tp-1 DBpedia,37 (0-5), tp-2 DBpedia,3 (25-30), tp-3 NYT,3 (54-59), 119-124</p> <p>Sum-Greedy: tp-1 DBpedia,37 (0-5), tp-2 DBpedia,3 (25-30), tp-3 NYT,3 (54-59), 119-124</p>

Continued on next page

Table B.35 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#8	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#9	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
	<p>CostFed</p> <p>Cost</p>
#10	Continued on next page

Table B.35 – continued from previous page

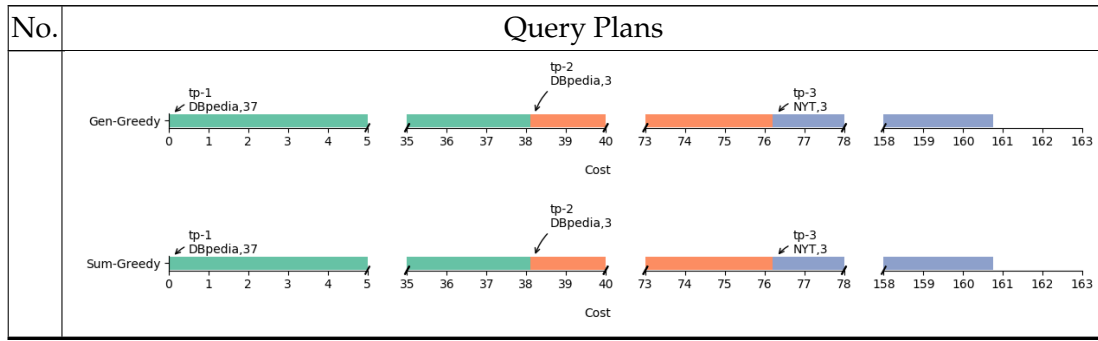
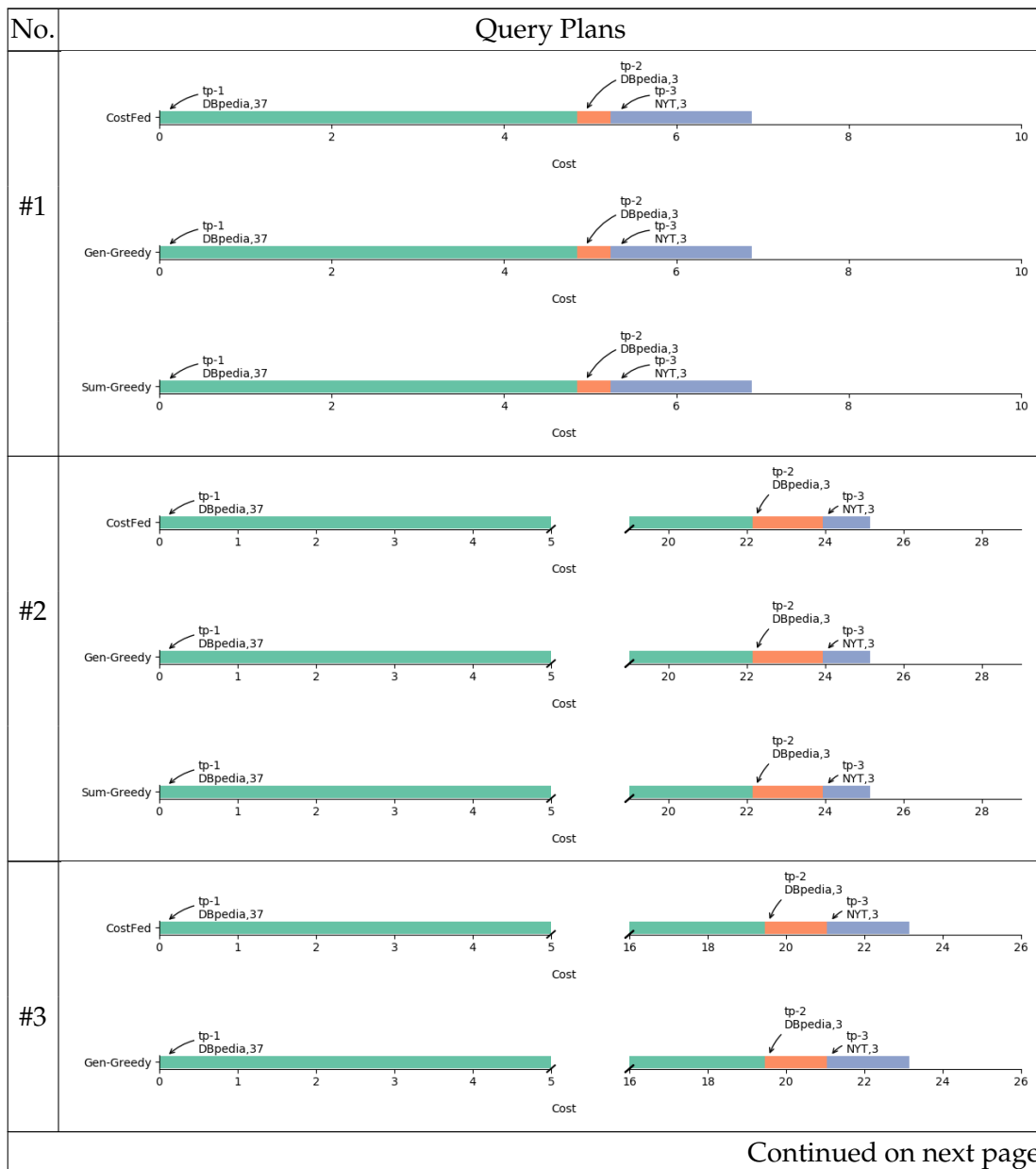


TABLE B.36: Query Plans of Query LD10 at Settings of 10 per Pricing Functions



Continued on next page

Table B.36 – continued from previous page


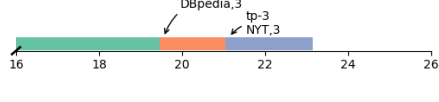
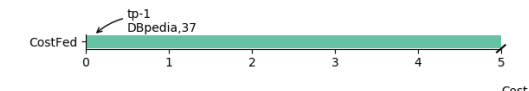
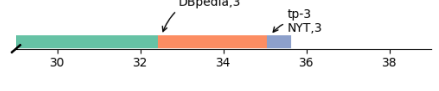
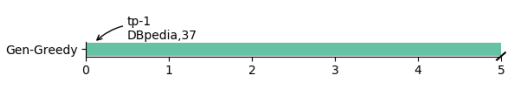
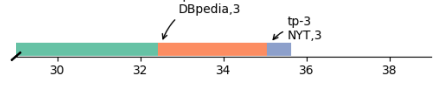
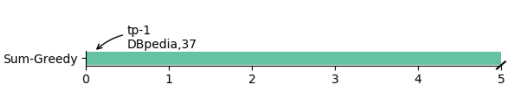
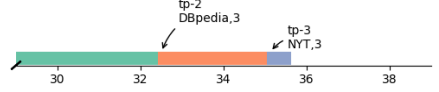
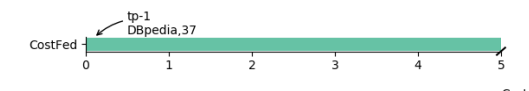
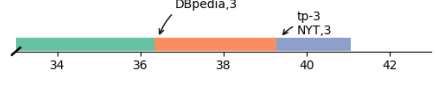
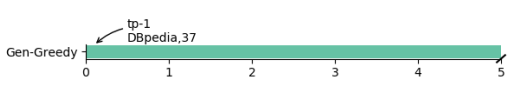
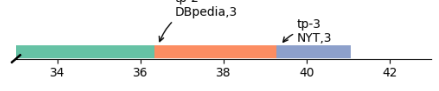
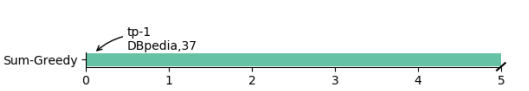
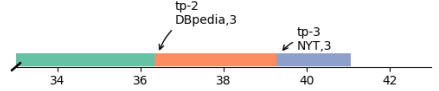
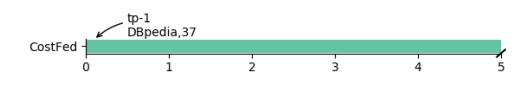
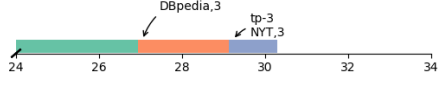
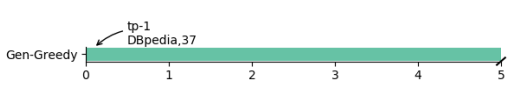
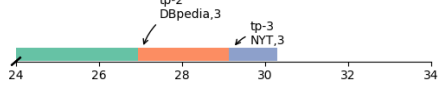
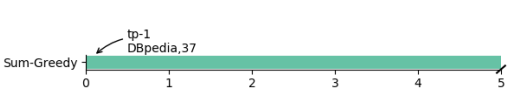
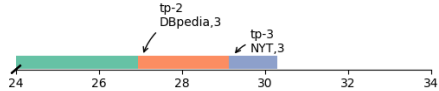
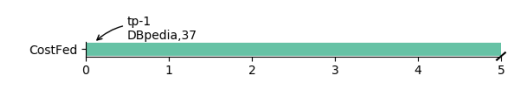
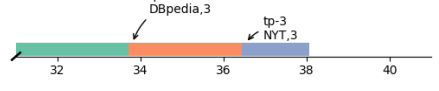
No.	Query Plans	
		
#4		
		
		
#5		
		
		
#6		
		
		
		
#7	Continued on next page	

Table B.36 – continued from previous page

No.	Query Plans	
	<div> <div> Gen-Greedy </div> <div> Sum-Greedy </div> </div>	
#8	<div> <div> CostFed </div> <div> Gen-Greedy </div> <div> Sum-Greedy </div> </div>	
#9	<div> <div> CostFed </div> <div> Gen-Greedy </div> <div> Sum-Greedy </div> </div>	
#10	<div> <div> CostFed </div> <div> Gen-Greedy </div> <div> Sum-Greedy </div> </div>	

TABLE B.37: Query Plans of Query LD11 at Settings of 10 Flat Pricing Functions

No.	Query Plans
#1	<p>CostFed: tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (66-70), tp-4 DBpedia,380 (134-138), tp-2 DBpedia,208 (202-206), tp-5 DBpedia,167 (270-274). Total Cost: 342.</p> <p>Gen-Greedy: tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (66-70), tp-4 DBpedia,380 (134-138), tp-2 DBpedia,208 (202-206), tp-5 DBpedia,167 (270-274). Total Cost: 342.</p> <p>Sum-Greedy: tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (66-70), tp-4 DBpedia,380 (134-138), tp-2 DBpedia,208 (202-206), tp-5 DBpedia,167 (270-274). Total Cost: 342.</p>
#2	<p>CostFed: tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (18-22), tp-4 DBpedia,380 (40-44), tp-2 DBpedia,208 (60-64), tp-5 DBpedia,167 (82-86). Total Cost: 108.</p> <p>Gen-Greedy: tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (18-22), tp-4 DBpedia,380 (40-44), tp-2 DBpedia,208 (60-64), tp-5 DBpedia,167 (82-86). Total Cost: 108.</p> <p>Sum-Greedy: tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (18-22), tp-4 DBpedia,380 (40-44), tp-2 DBpedia,208 (60-64), tp-5 DBpedia,167 (82-86). Total Cost: 108.</p>
#3	<p>CostFed: tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (20-24), tp-4 DBpedia,380 (42-46), tp-2 DBpedia,208 (64-68), tp-5 DBpedia,167 (84-88). Total Cost: 110.</p> <p>Gen-Greedy: tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (20-24), tp-4 DBpedia,380 (42-46), tp-2 DBpedia,208 (64-68), tp-5 DBpedia,167 (84-88). Total Cost: 110.</p> <p>Sum-Greedy: tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (20-24), tp-4 DBpedia,380 (42-46), tp-2 DBpedia,208 (64-68), tp-5 DBpedia,167 (84-88). Total Cost: 110.</p>
#4	<p>CostFed: tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (92-96), tp-4 DBpedia,380 (186-190), tp-2 DBpedia,208 (280-284), tp-5 DBpedia,167 (376-380). Total Cost: 472.</p> <p>Gen-Greedy: tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (92-96), tp-4 DBpedia,380 (186-190), tp-2 DBpedia,208 (280-284), tp-5 DBpedia,167 (376-380). Total Cost: 472.</p>

Continued on next page

Table B.37 – continued from previous page

No.	Query Plans
#5	
#6	
#7	
#8	
#8	Continued on next page

Table B.37 – continued from previous page

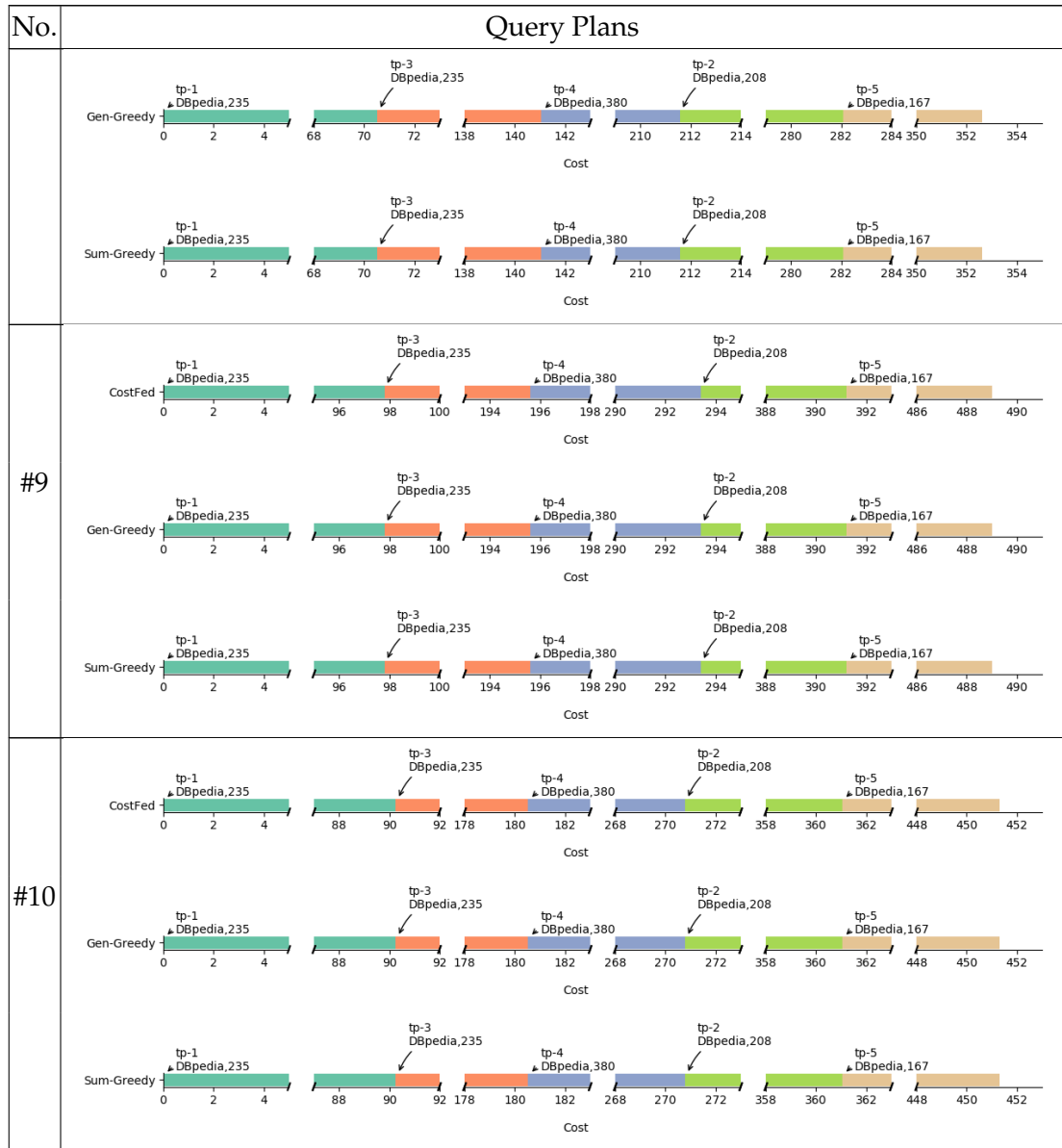
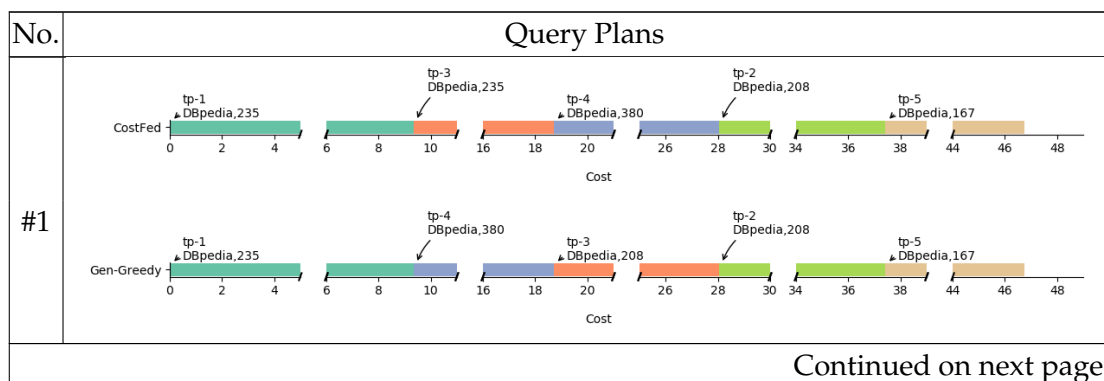


TABLE B.38: Query Plans of Query LD11 at Settings of 10 freemium Pricing Functions



Continued on next page

Table B.38 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#2	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#3	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#4	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#5	<p>CostFed</p> <p>Cost</p>

Continued on next page

Table B.38 – continued from previous page

No.	Query Plans
	<p>The first section displays query plans for two algorithms: Gen-Greedy and Sum-Greedy. Each algorithm has three horizontal bars representing different task partitions: tp-1 DBpedia,235 (green), tp-3 DBpedia,235 (orange), and tp-4 DBpedia,380 (blue). The x-axis is labeled 'Cost' and ranges from 0 to 390. The Gen-Greedy plan shows a total cost of 390, while the Sum-Greedy plan shows a total cost of 380.</p>
#6	<p>Section #6 shows query plans for three algorithms: CostFed, Gen-Greedy, and Sum-Greedy. Each algorithm has three horizontal bars representing different task partitions: tp-1 DBpedia,235 (green), tp-3 DBpedia,235 (orange), and tp-4 DBpedia,380 (blue). The x-axis is labeled 'Cost' and ranges from 0 to 144. The CostFed plan shows a total cost of 144, while the Gen-Greedy and Sum-Greedy plans show a total cost of 140.</p>
#7	<p>Section #7 shows query plans for three algorithms: CostFed, Gen-Greedy, and Sum-Greedy. Each algorithm has three horizontal bars representing different task partitions: tp-1 DBpedia,235 (green), tp-3 DBpedia,235 (orange), and tp-4 DBpedia,380 (blue). The x-axis is labeled 'Cost' and ranges from 0 to 208. The CostFed plan shows a total cost of 208, while the Gen-Greedy and Sum-Greedy plans show a total cost of 204.</p>
#8	<p>Section #8 shows query plans for three algorithms: CostFed, Gen-Greedy, and Sum-Greedy. Each algorithm has three horizontal bars representing different task partitions: tp-1 DBpedia,235 (green), tp-3 DBpedia,235 (orange), and tp-4 DBpedia,380 (blue). The x-axis is labeled 'Cost' and ranges from 0 to 120. The CostFed plan shows a total cost of 120, while the Gen-Greedy and Sum-Greedy plans show a total cost of 116.</p>
Continued on next page	

Table B.38 – continued from previous page

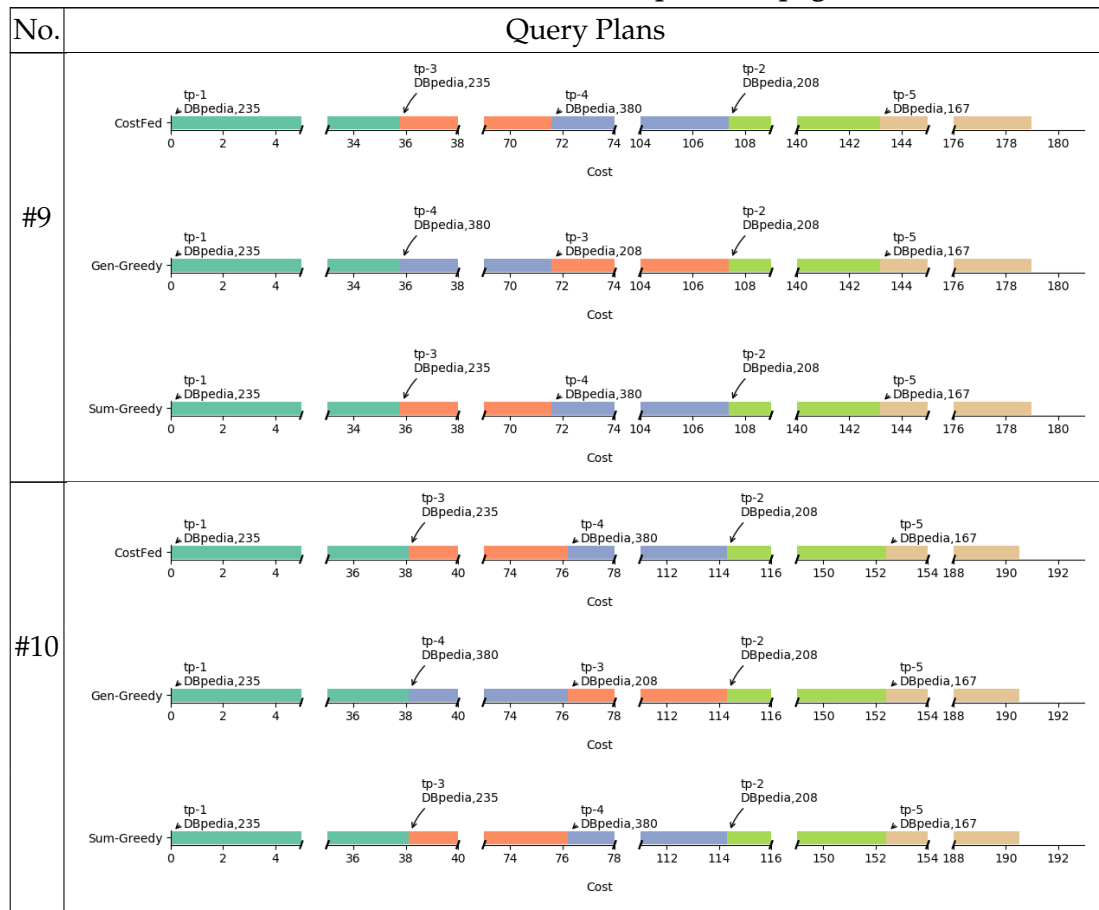
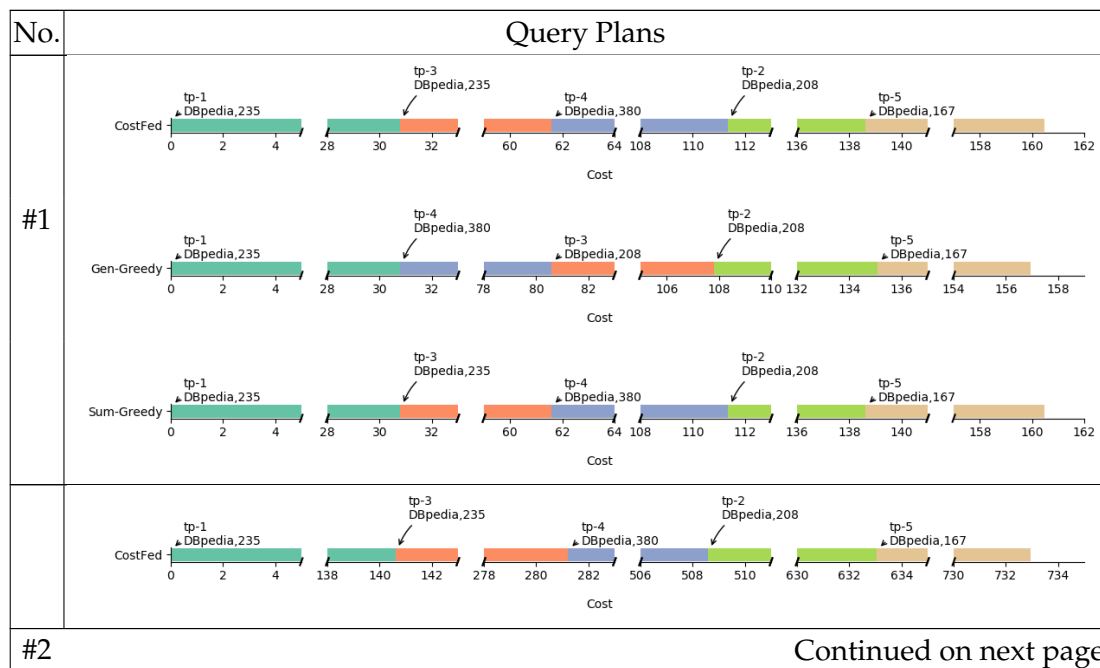


TABLE B.39: Query Plans of Query LD11 at Settings of 10 per Pricing Functions



Continued on next page

Table B.39 – continued from previous page

No.	Query Plans
	<p>Gen-Greedy</p> <p>tp-1 DBpedia,235 (0-4), tp-4 DBpedia,380 (138-140), tp-3 DBpedia,208 (366-370), tp-2 DBpedia,208 (490-492), tp-5 DBpedia,167 (614-616)</p> <p>Cost: 0, 2, 4, 138, 140, 142, 366, 368, 370, 490, 492, 494, 614, 616, 618, 714, 716, 718</p> <p>Sum-Greedy</p> <p>tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (138-140), tp-4 DBpedia,380 (278-282), tp-2 DBpedia,208 (506-510), tp-5 DBpedia,167 (630-632)</p> <p>Cost: 0, 2, 4, 138, 140, 142, 278, 280, 282, 506, 508, 510, 630, 632, 634, 730, 732, 734</p>
#3	<p>CostFed</p> <p>tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (122-124), tp-4 DBpedia,380 (244-246), tp-2 DBpedia,208 (444-446), tp-5 DBpedia,167 (554-556)</p> <p>Cost: 0, 2, 4, 122, 124, 126, 244, 246, 248, 444, 446, 448, 554, 556, 558, 642, 644, 646</p> <p>Gen-Greedy</p> <p>tp-1 DBpedia,235 (0-4), tp-4 DBpedia,380 (122-124), tp-3 DBpedia,208 (320-322), tp-2 DBpedia,208 (430-432), tp-5 DBpedia,167 (540-542)</p> <p>Cost: 0, 2, 4, 122, 124, 126, 320, 322, 324, 430, 432, 434, 540, 542, 544, 628, 630, 632</p> <p>Sum-Greedy</p> <p>tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (122-124), tp-4 DBpedia,380 (244-246), tp-2 DBpedia,208 (444-446), tp-5 DBpedia,167 (554-556)</p> <p>Cost: 0, 2, 4, 122, 124, 126, 244, 246, 248, 444, 446, 448, 554, 556, 558, 642, 644, 646</p>
#4	<p>CostFed</p> <p>tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (204-206), tp-4 DBpedia,380 (410-412), tp-2 DBpedia,208 (742-744), tp-5 DBpedia,167 (924-926)</p> <p>Cost: 0, 2, 4, 204, 206, 208, 410, 412, 414, 742, 744, 746, 924, 926, 928, 1070, 1072, 1074</p> <p>Gen-Greedy</p> <p>tp-1 DBpedia,235 (0-4), tp-4 DBpedia,380 (204-206), tp-3 DBpedia,208 (536-538), tp-2 DBpedia,208 (718-720), tp-5 DBpedia,167 (900-902)</p> <p>Cost: 0, 2, 4, 204, 206, 208, 536, 538, 540, 718, 720, 722, 900, 902, 904, 1046, 1048, 1050</p> <p>Sum-Greedy</p> <p>tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (204-206), tp-2 DBpedia,235 (410-412), tp-4 DBpedia,380 (614-616), tp-5 DBpedia,167 (948-950)</p> <p>Cost: 0, 2, 4, 204, 206, 208, 410, 412, 414, 614, 616, 618, 948, 950, 952, 1094, 1096, 1098</p>
#5	<p>CostFed</p> <p>tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (228-230), tp-4 DBpedia,380 (458-460), tp-2 DBpedia,208 (832-834), tp-5 DBpedia,167 (1036-1038)</p> <p>Cost: 0, 2, 4, 228, 230, 232, 458, 460, 462, 832, 834, 836, 1036, 1038, 1040, 1200, 1202, 1204</p> <p>Gen-Greedy</p> <p>tp-1 DBpedia,235 (0-4), tp-4 DBpedia,380 (228-230), tp-3 DBpedia,208 (602-604), tp-2 DBpedia,208 (806-808), tp-5 DBpedia,167 (1010-1012)</p> <p>Cost: 0, 2, 4, 228, 230, 232, 602, 604, 606, 806, 808, 810, 1010, 1012, 1014, 1174, 1176, 1178</p> <p>Sum-Greedy</p> <p>tp-1 DBpedia,235 (0-4), tp-3 DBpedia,235 (228-230), tp-2 DBpedia,235 (458-460), tp-4 DBpedia,380 (690-692), tp-5 DBpedia,167 (1062-1064)</p> <p>Cost: 0, 2, 4, 228, 230, 232, 458, 460, 462, 690, 692, 694, 1062, 1064, 1066, 1226, 1228, 1230</p>

Continued on next page

Table B.39 – continued from previous page

No.	Query Plans
#6	<p>CostFed</p> <p>Cost: 0 2 4 168 170 172 340 342 344 618 620 768 770 772 890 892 894</p>
	<p>Gen-Greedy</p> <p>Cost: 0 2 4 168 170 172 446 448 450 598 600 748 750 752 870 872 874</p>
	<p>Sum-Greedy</p> <p>Cost: 0 2 4 168 170 172 340 342 344 618 620 768 770 772 890 892 894</p>
#7	<p>CostFed</p> <p>Cost: 0 2 4 212 214 216 426 428 430 772 774 776 962 964 966 1114 1116 1118</p>
	<p>Gen-Greedy</p> <p>Cost: 0 2 4 212 214 216 558 560 562 748 750 752 936 938 940 1088 1090 1092</p>
	<p>Sum-Greedy</p> <p>Cost: 0 2 4 212 214 216 426 428 430 772 774 776 962 964 966 1114 1116 1118</p>
#8	<p>CostFed</p> <p>Cost: 0 2 4 112 114 116 226 228 230 410 412 414 510 512 514 592 594 596</p>
	<p>Gen-Greedy</p> <p>Cost: 0 2 4 112 114 116 296 298 300 396 398 400 498 500 502 578 580 582</p>
	<p>Sum-Greedy</p> <p>Cost: 0 2 4 112 114 116 226 228 230 410 412 414 510 512 514 592 594 596</p>
#9	<p>CostFed</p> <p>Cost: 0 2 4 200 202 204 402 404 406 728 730 732 908 910 912 1052 1054 1056</p>
	<p>Gen-Greedy</p> <p>Cost: 0 2 4 200 202 204 526 528 530 706 708 710 884 886 888 1028 1030 1032</p>

Continued on next page

Table B.39 – continued from previous page

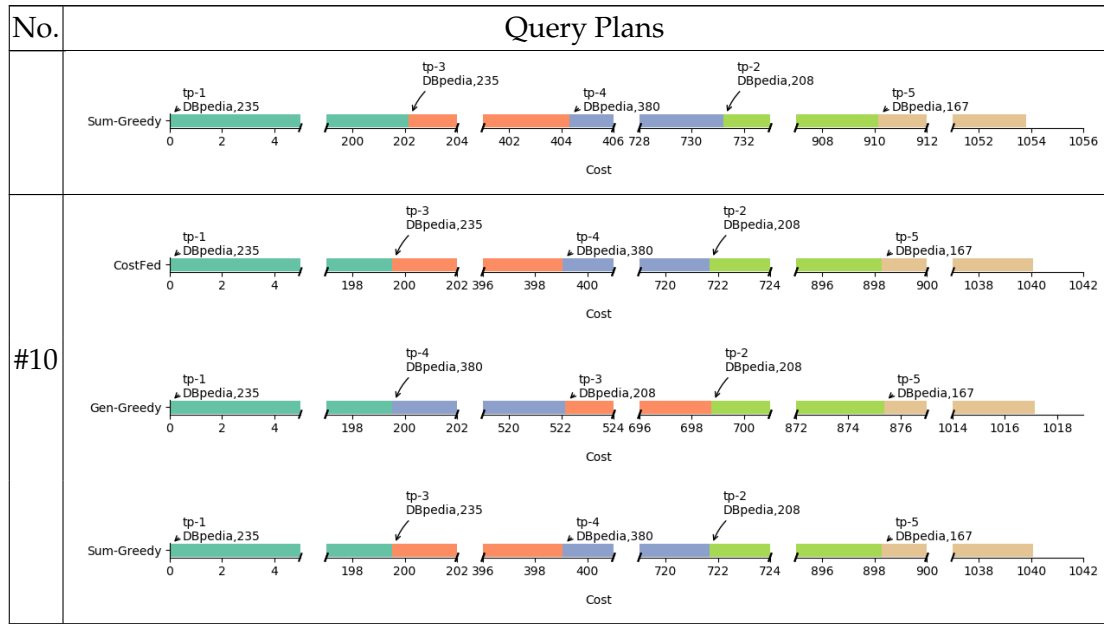
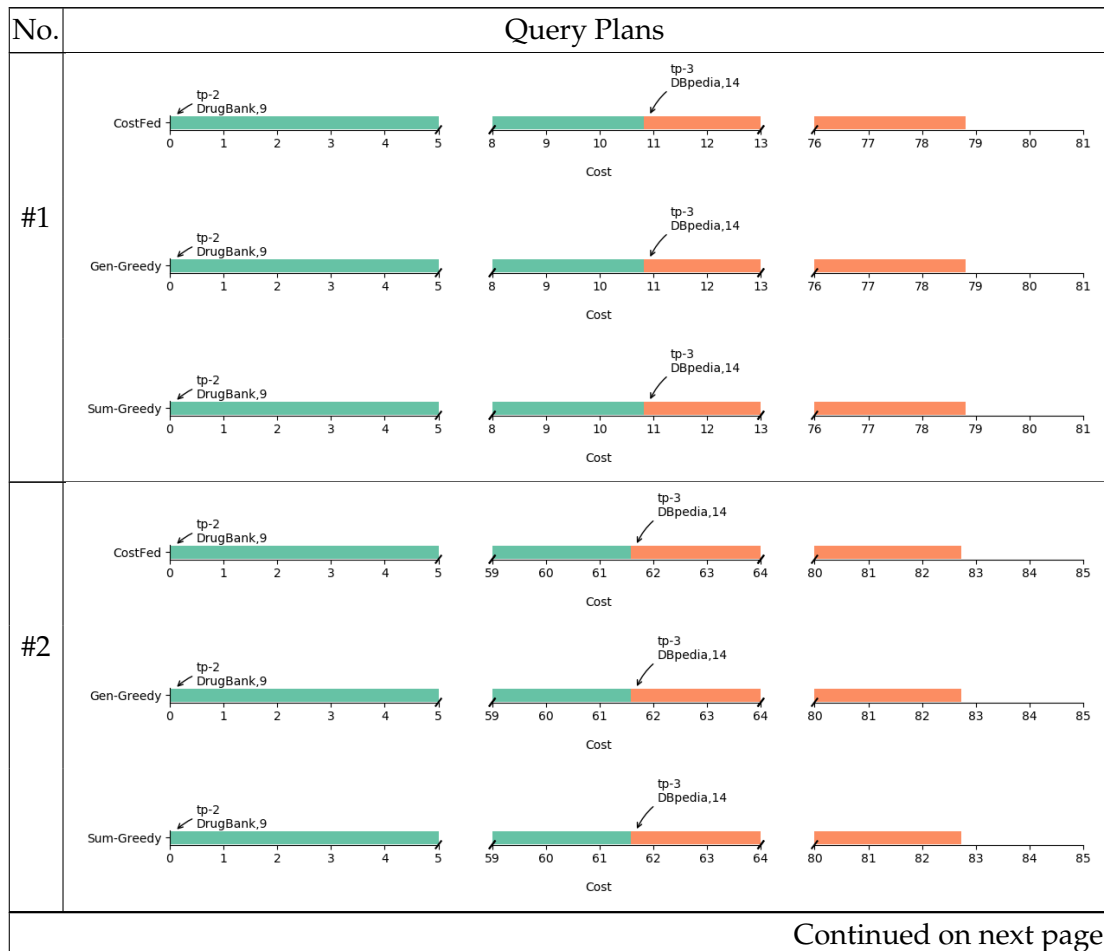


TABLE B.40: Query Plans of Query LS2 at Settings of 10 Flat Pricing Functions



Continued on next page

Table B.40 – continued from previous page

No.	Query Plans
#3	
#4	
#5	
#6	

Continued on next page

Table B.40 – continued from previous page

No.	Query Plans
#7	
#8	
#9	
#10	

Continued on next page

Table B.40 – continued from previous page

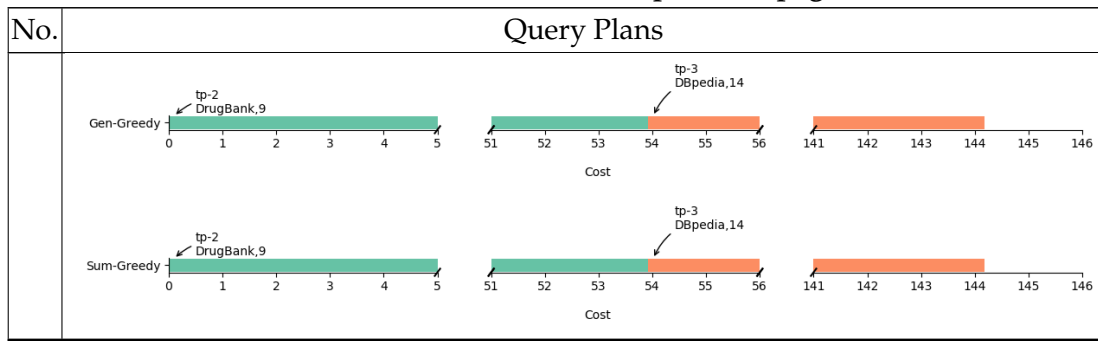
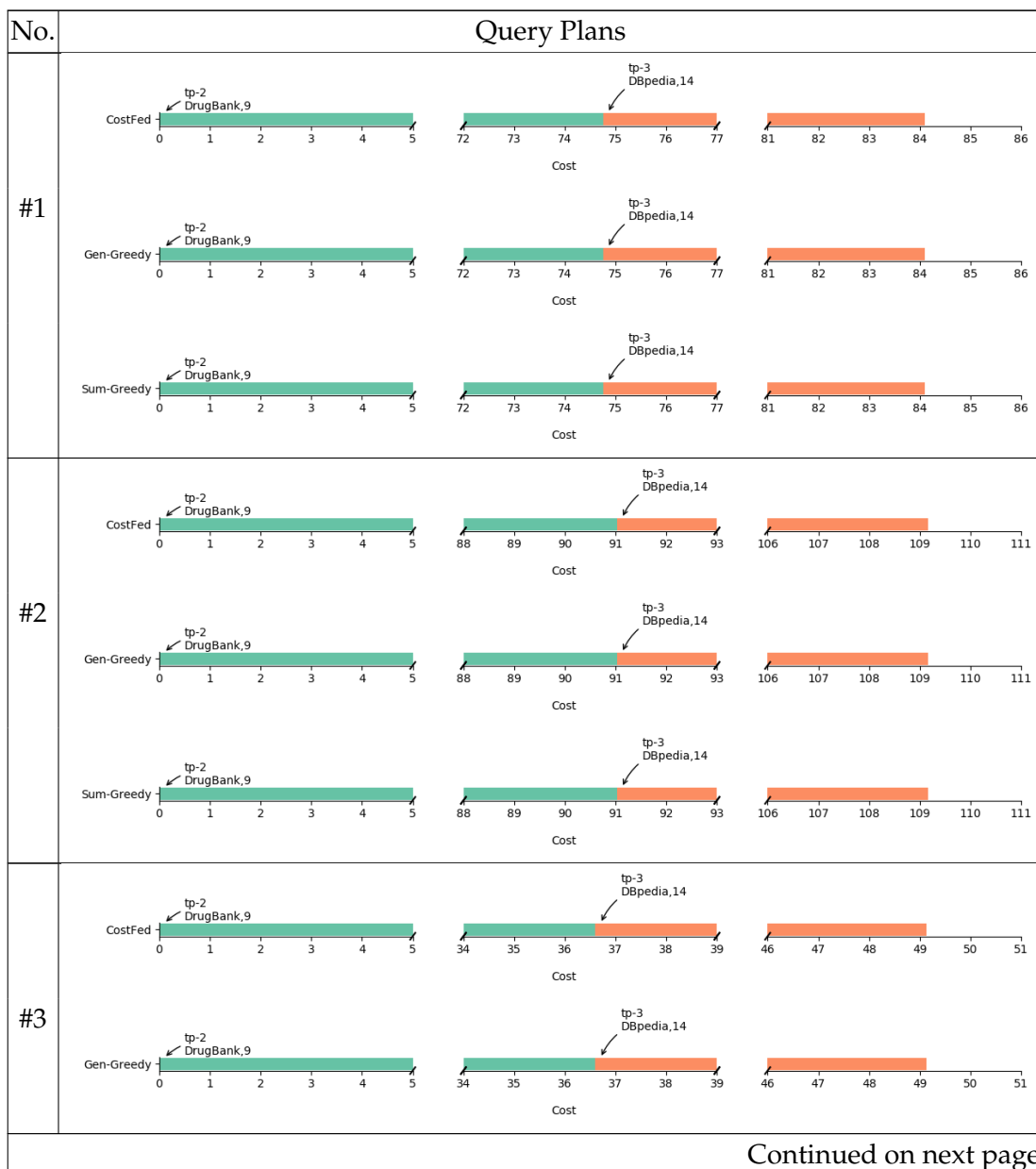


TABLE B.41: Query Plans of Query LS2 at Settings of 10 freemium Pricing Functions



Continued on next page

Table B.41 – continued from previous page

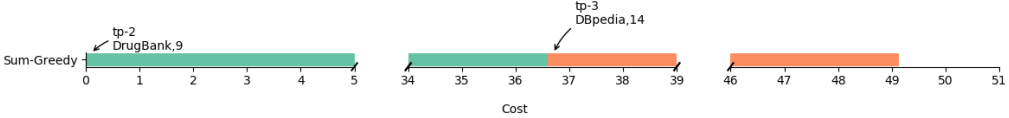
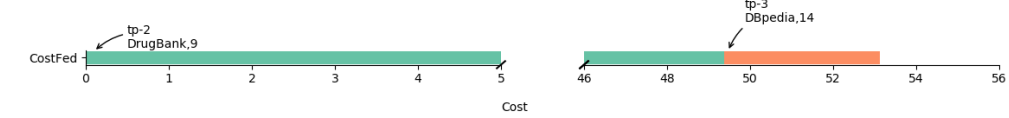
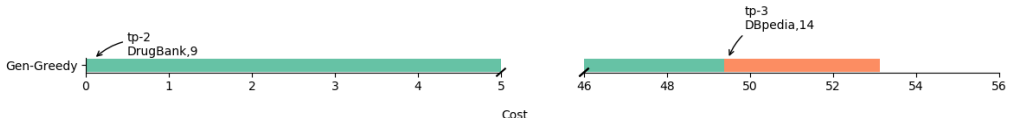
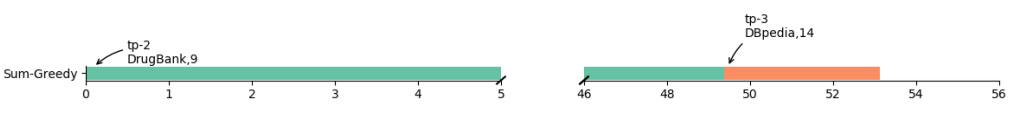
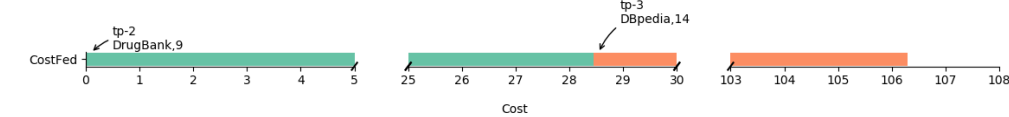
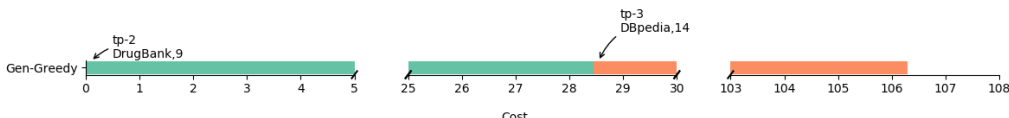
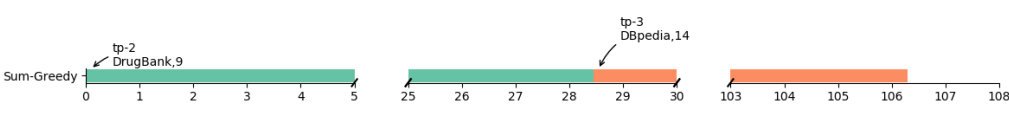
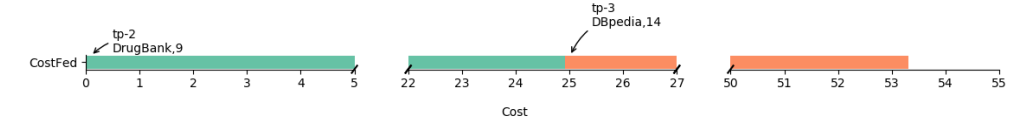
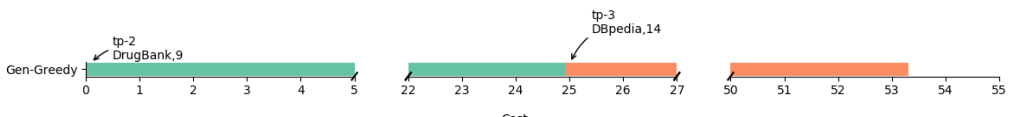
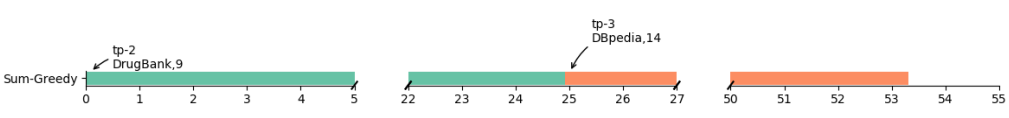
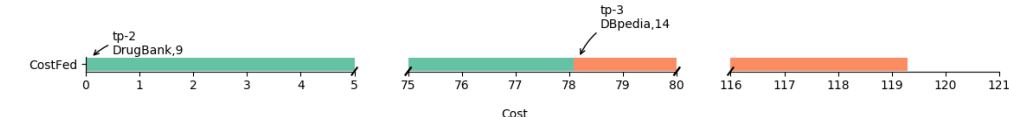
No.	Query Plans
	
#4	  
#5	  
#6	  
	
#7	Continued on next page

Table B.41 – continued from previous page

No.	Query Plans		
	Gen-Greedy tp-2 DrugBank,9	tp-3 DBpedia,14 Cost	
	Sum-Greedy tp-2 DrugBank,9	tp-3 DBpedia,14 Cost	
	CostFed tp-2 DrugBank,9	tp-3 DBpedia,14 Cost	
#8	Gen-Greedy tp-2 DrugBank,9	tp-3 DBpedia,14 Cost	
	Sum-Greedy tp-2 DrugBank,9	tp-3 DBpedia,14 Cost	
	CostFed tp-2 DrugBank,9	tp-3 DBpedia,14 Cost	
#9	Gen-Greedy tp-2 DrugBank,9	tp-3 DBpedia,14 Cost	
	Sum-Greedy tp-2 DrugBank,9	tp-3 DBpedia,14 Cost	
	CostFed tp-2 DrugBank,9	tp-3 DBpedia,14 Cost	
#10	Gen-Greedy tp-2 DrugBank,9	tp-3 DBpedia,14 Cost	
	Sum-Greedy tp-2 DrugBank,9	tp-3 DBpedia,14 Cost	
	CostFed tp-2 DrugBank,9	tp-3 DBpedia,14 Cost	

TABLE B.42: Query Plans of Query LS2 at Settings of 10 per Pricing Functions

No.	Query Plans
#1	<p>CostFed: tp-2 DrugBank,9 (0-5), tp-3 DBpedia,14 (8-10)</p> <p>Gen-Greedy: tp-2 DrugBank,9 (0-5), tp-3 DBpedia,14 (8-10)</p> <p>Sum-Greedy: tp-2 DrugBank,9 (0-5), tp-3 DBpedia,14 (8-10)</p>
#2	<p>CostFed: tp-2 DrugBank,9 (0-5), tp-3 DBpedia,14 (5-10)</p> <p>Gen-Greedy: tp-2 DrugBank,9 (0-5), tp-3 DBpedia,14 (5-10)</p> <p>Sum-Greedy: tp-2 DrugBank,9 (0-5), tp-3 DBpedia,14 (5-10)</p>
#3	<p>CostFed: tp-2 DrugBank,9 (0-7), tp-3 DBpedia,14 (7-10)</p> <p>Gen-Greedy: tp-2 DrugBank,9 (0-7), tp-3 DBpedia,14 (7-10)</p> <p>Sum-Greedy: tp-2 DrugBank,9 (0-7), tp-3 DBpedia,14 (7-10)</p>
#4	<p>CostFed: tp-2 DrugBank,9 (0-1), tp-3 DBpedia,14 (1-5)</p> <p>Gen-Greedy: tp-2 DrugBank,9 (0-1), tp-3 DBpedia,14 (1-5)</p>
Continued on next page	

Table B.42 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#5	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#6	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
	<p>CostFed</p> <p>Cost</p>
#8	Continued on next page

Table B.42 – continued from previous page

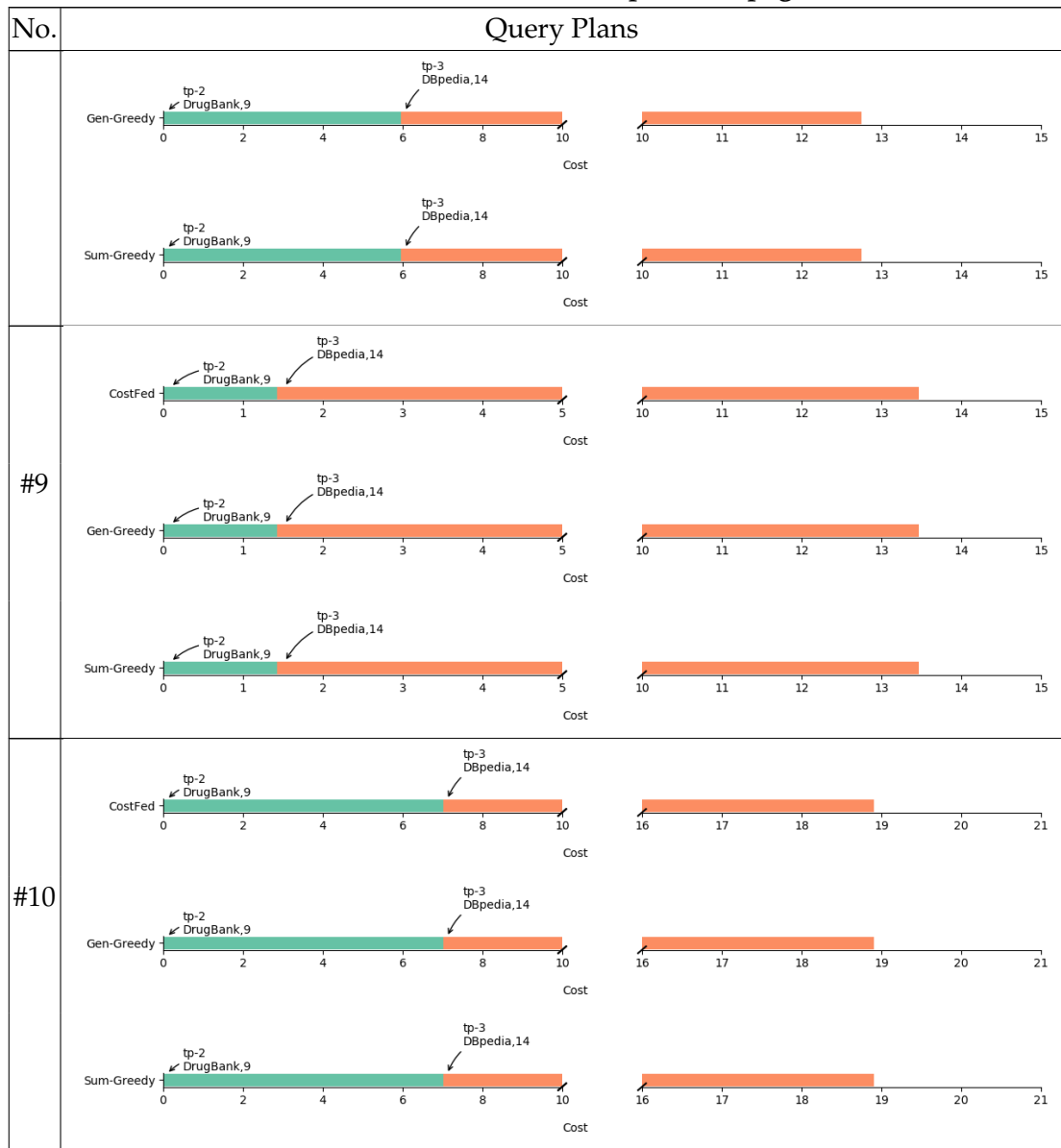


TABLE B.43: Query Plans of Query LS3 at Settings of 10 Flat Pricing Functions

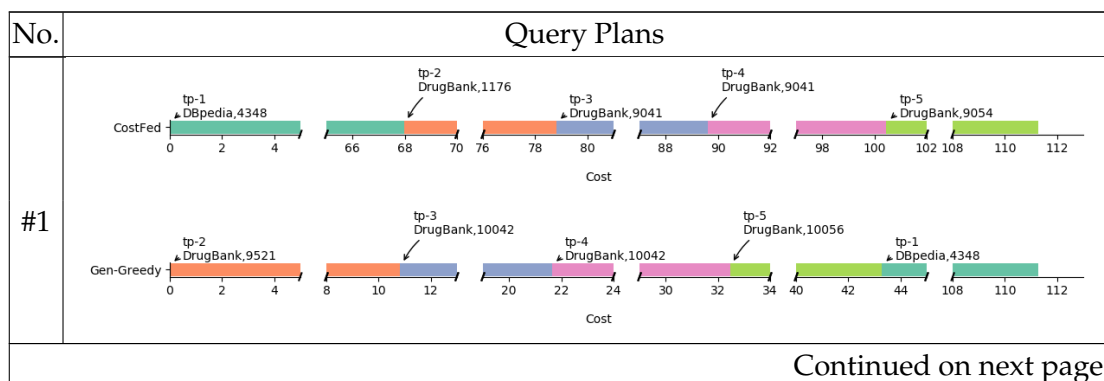


Table B.43 – continued from previous page

No.	Query Plans
#2	<p>Sum-Greedy</p> <p>Cost</p>
	<p>CostFed</p> <p>Cost</p>
	<p>Gen-Greedy</p> <p>Cost</p>
#3	<p>Sum-Greedy</p> <p>Cost</p>
	<p>Gen-Greedy</p> <p>Cost</p>
	<p>CostFed</p> <p>Cost</p>
#4	<p>Sum-Greedy</p> <p>Cost</p>
	<p>Gen-Greedy</p> <p>Cost</p>
	<p>CostFed</p> <p>Cost</p>
#5	Continued on next page

Table B.43 – continued from previous page

No.	Query Plans
	<p>Gen-Greedy: tp-1 DBpedia,4348 (0-4), tp-2 DrugBank,1176 (4-66), tp-3 DrugBank,9041 (66-132), tp-4 DrugBank,9041 (132-198), tp-5 DrugBank,9054 (198-268). Cost: 268.</p> <p>Sum-Greedy: tp-1 DBpedia,4348 (0-4), tp-2 DrugBank,1176 (4-66), tp-3 DrugBank,9041 (66-132), tp-4 DrugBank,9041 (132-198), tp-5 DrugBank,9054 (198-268). Cost: 268.</p>
#6	<p>CostFed: tp-1 DBpedia,4348 (0-40), tp-2 DrugBank,1176 (40-78), tp-3 DrugBank,9041 (78-116), tp-4 DrugBank,9041 (116-154), tp-5 DrugBank,9054 (154-196). Cost: 196.</p> <p>Gen-Greedy: tp-2 DrugBank,9521 (0-36), tp-3 DrugBank,10042 (36-74), tp-4 DrugBank,10042 (74-112), tp-5 DrugBank,10056 (112-148), tp-1 DBpedia,4348 (148-196). Cost: 196.</p> <p>Sum-Greedy: tp-2 DrugBank,9521 (0-36), tp-1 DBpedia,4348 (36-78), tp-3 DrugBank,9041 (78-116), tp-4 DrugBank,9041 (116-154), tp-5 DrugBank,9054 (154-196). Cost: 196.</p>
#7	<p>CostFed: tp-1 DBpedia,4348 (0-30), tp-2 DrugBank,1176 (30-36), tp-3 DrugBank,9041 (36-42), tp-4 DrugBank,9041 (42-50), tp-5 DrugBank,9054 (50-60). Cost: 60.</p> <p>Gen-Greedy: tp-2 DrugBank,9521 (0.0-10.0), tp-3 DrugBank,10042 (10-14), tp-4 DrugBank,10042 (14-18), tp-5 DrugBank,10056 (18-24), tp-1 DBpedia,4348 (24-60). Cost: 60.</p> <p>Sum-Greedy: tp-2 DrugBank,9521 (0.0-10.0), tp-1 DBpedia,4348 (10-36), tp-3 DrugBank,9041 (36-42), tp-4 DrugBank,9041 (42-50), tp-5 DrugBank,9054 (50-60). Cost: 60.</p>
#8	<p>CostFed: tp-1 DBpedia,4348 (0-68), tp-2 DrugBank,1176 (68-158), tp-3 DrugBank,9041 (158-248), tp-4 DrugBank,9041 (248-336), tp-5 DrugBank,9054 (336-428). Cost: 428.</p> <p>Gen-Greedy: tp-1 DBpedia,4348 (0-68), tp-2 DrugBank,1176 (68-158), tp-3 DrugBank,9041 (158-248), tp-4 DrugBank,9041 (248-336), tp-5 DrugBank,9054 (336-428). Cost: 428.</p> <p>Sum-Greedy: tp-1 DBpedia,4348 (0-68), tp-2 DrugBank,1176 (68-158), tp-3 DrugBank,9041 (158-248), tp-4 DrugBank,9041 (248-336), tp-5 DrugBank,9054 (336-428). Cost: 428.</p>
Continued on next page	

Table B.43 – continued from previous page

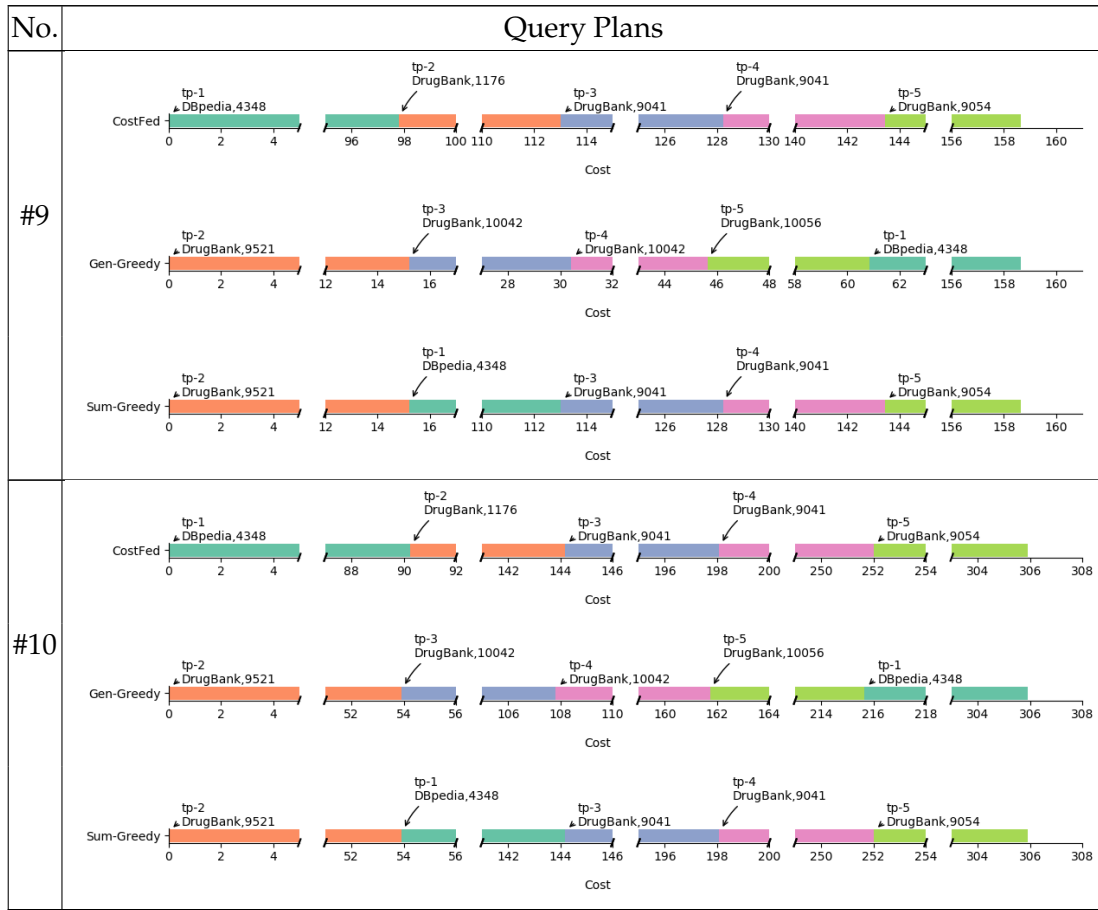
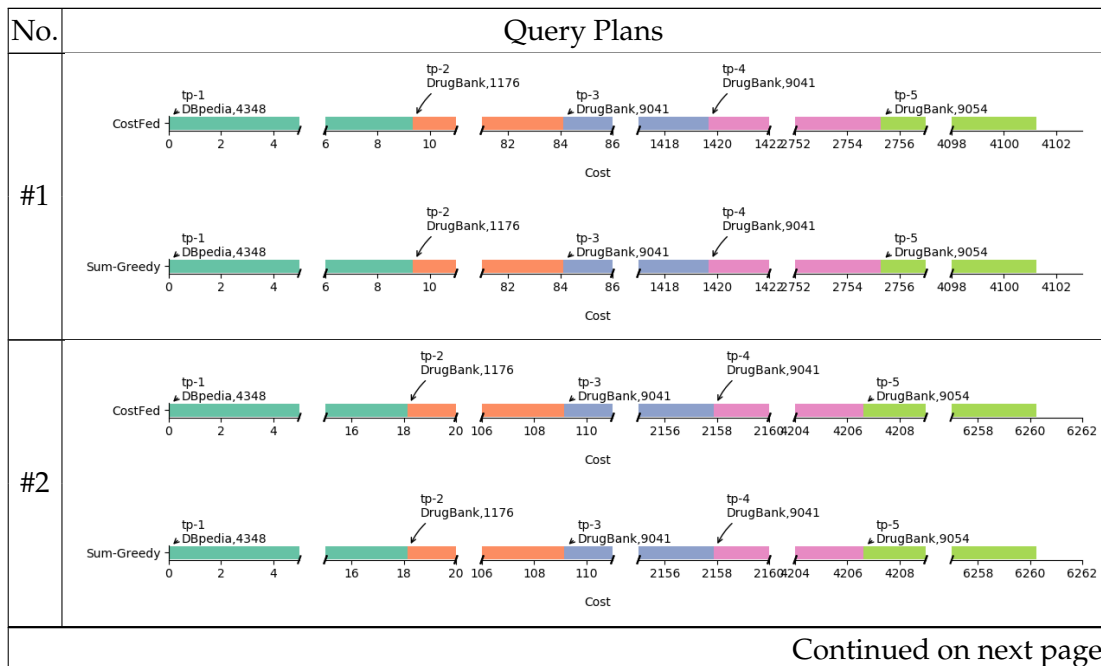


TABLE B.44: Query Plans of Query LS3 at Settings of 10 freemium Pricing Functions



Continued on next page

Table B.44 – continued from previous page

No.	Query Plans
#3	<p>CostFed: tp-1 DBpedia,4348 (0-4), tp-2 DrugBank,1176 (10-12), tp-3 DrugBank,9041 (46-50), tp-4 DrugBank,9041 (632-634), tp-5 DrugBank,9054 (1218-1222), 1814</p> <p>Sum-Greedy: tp-1 DBpedia,4348 (0-4), tp-2 DrugBank,1176 (10-12), tp-3 DrugBank,9041 (46-50), tp-4 DrugBank,9041 (632-634), tp-5 DrugBank,9054 (1218-1222), 1814</p>
#4	<p>CostFed: tp-1 DBpedia,4348 (0-4), tp-2 DrugBank,1176 (4218-4220), tp-3 DrugBank,9041 (4268-4270), tp-4 DrugBank,9041 (4272-4274), tp-5 DrugBank,9054 (9950-9952), 1.28e4</p> <p>Sum-Greedy: tp-2 DrugBank,9521 (0-4), tp-1 DBpedia,4348 (3166-3168), tp-3 DrugBank,9041 (3170-3172), tp-4 DrugBank,9041 (7388-7390), tp-5 DrugBank,9054 (10228-10230), 1.592e4</p>
#5	<p>CostFed: tp-1 DBpedia,4348 (0-4), tp-2 DrugBank,1176 (76-78), tp-3 DrugBank,9041 (104-106), tp-4 DrugBank,9041 (4534-4536), tp-5 DrugBank,9054 (8964-8966), 1.34e4</p> <p>Sum-Greedy: tp-1 DBpedia,4348 (0-4), tp-2 DrugBank,1176 (76-78), tp-3 DrugBank,9041 (104-106), tp-4 DrugBank,9041 (4534-4536), tp-5 DrugBank,9054 (8964-8966), 1.34e4</p>
#6	<p>CostFed: tp-1 DBpedia,4348 (0-4), tp-2 DrugBank,1176 (26-28), tp-3 DrugBank,9041 (886-888), tp-4 DrugBank,9041 (890-892), tp-5 DrugBank,9054 (8504-8506), 2.375e4</p> <p>Sum-Greedy: tp-1 DBpedia,4348 (0-4), tp-2 DrugBank,1176 (26-28), tp-3 DrugBank,9041 (886-888), tp-4 DrugBank,9041 (890-892), tp-5 DrugBank,9054 (8504-8506), 2.375e4</p>
#7	<p>CostFed: tp-1 DBpedia,4348 (0-4), tp-2 DrugBank,1176 (38-40), tp-3 DrugBank,9041 (116-118), tp-4 DrugBank,9041 (120-122), tp-5 DrugBank,9054 (264-266), 564</p> <p>Sum-Greedy: tp-1 DBpedia,4348 (0-4), tp-2 DrugBank,1176 (38-40), tp-3 DrugBank,9041 (116-118), tp-4 DrugBank,9041 (120-122), tp-5 DrugBank,9054 (264-266), 564</p>
#8	<p>CostFed: tp-1 DBpedia,4348 (0-4), tp-2 DrugBank,1176 (22-24), tp-3 DrugBank,9041 (100-102), tp-4 DrugBank,9041 (104-106), tp-5 DrugBank,9054 (6866-6868), 20410</p> <p>Sum-Greedy: tp-1 DBpedia,4348 (0-4), tp-2 DrugBank,1176 (22-24), tp-3 DrugBank,9041 (100-102), tp-4 DrugBank,9041 (104-106), tp-5 DrugBank,9054 (6866-6868), 20410</p>

Continued on next page

Table B.44 – continued from previous page

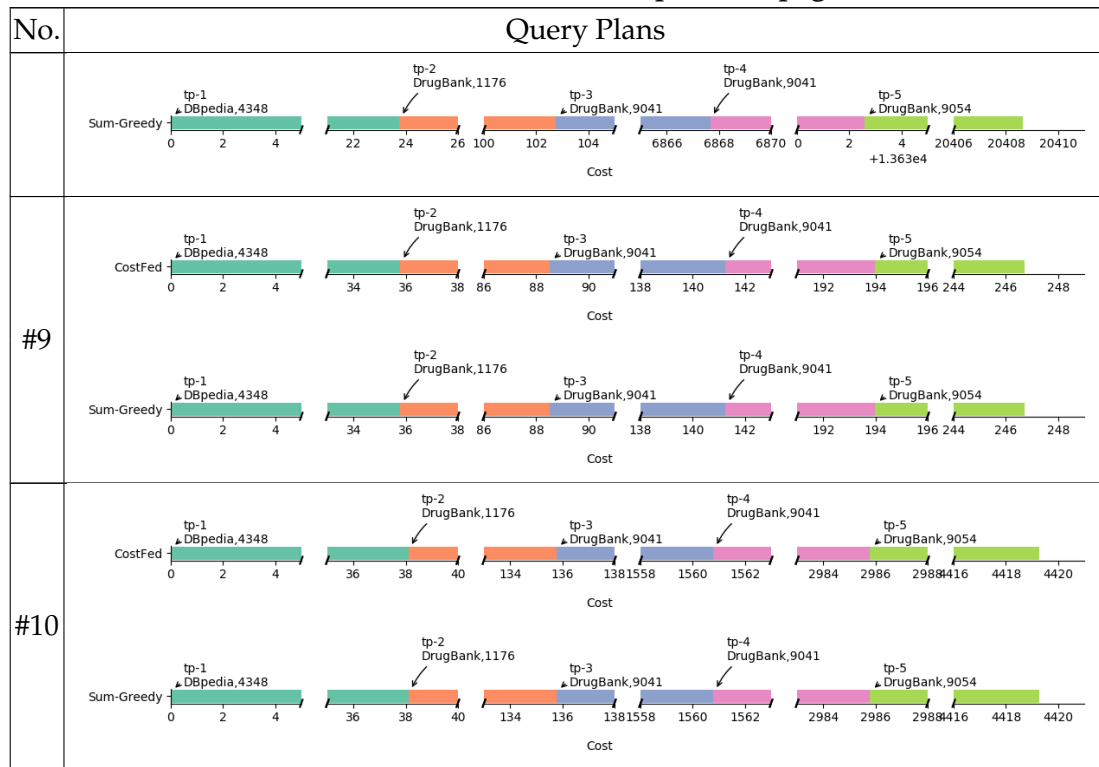
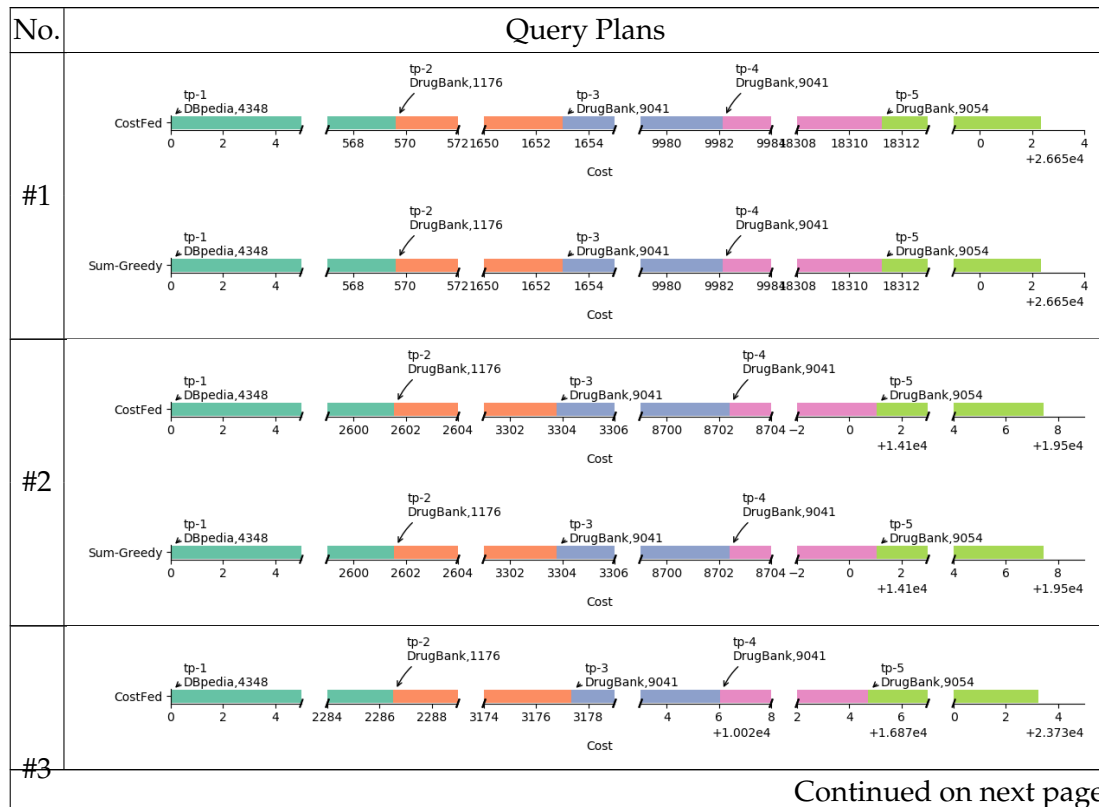


TABLE B.45: Query Plans of Query LS3 at Settings of 10 per Pricing Functions



Continued on next page

Table B.45 – continued from previous page

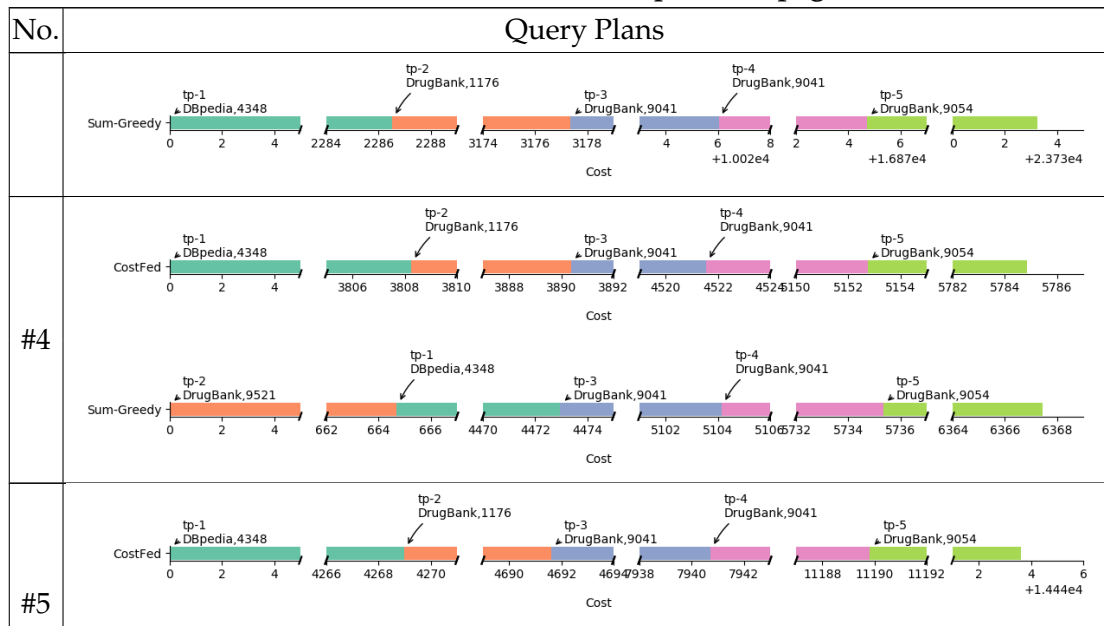
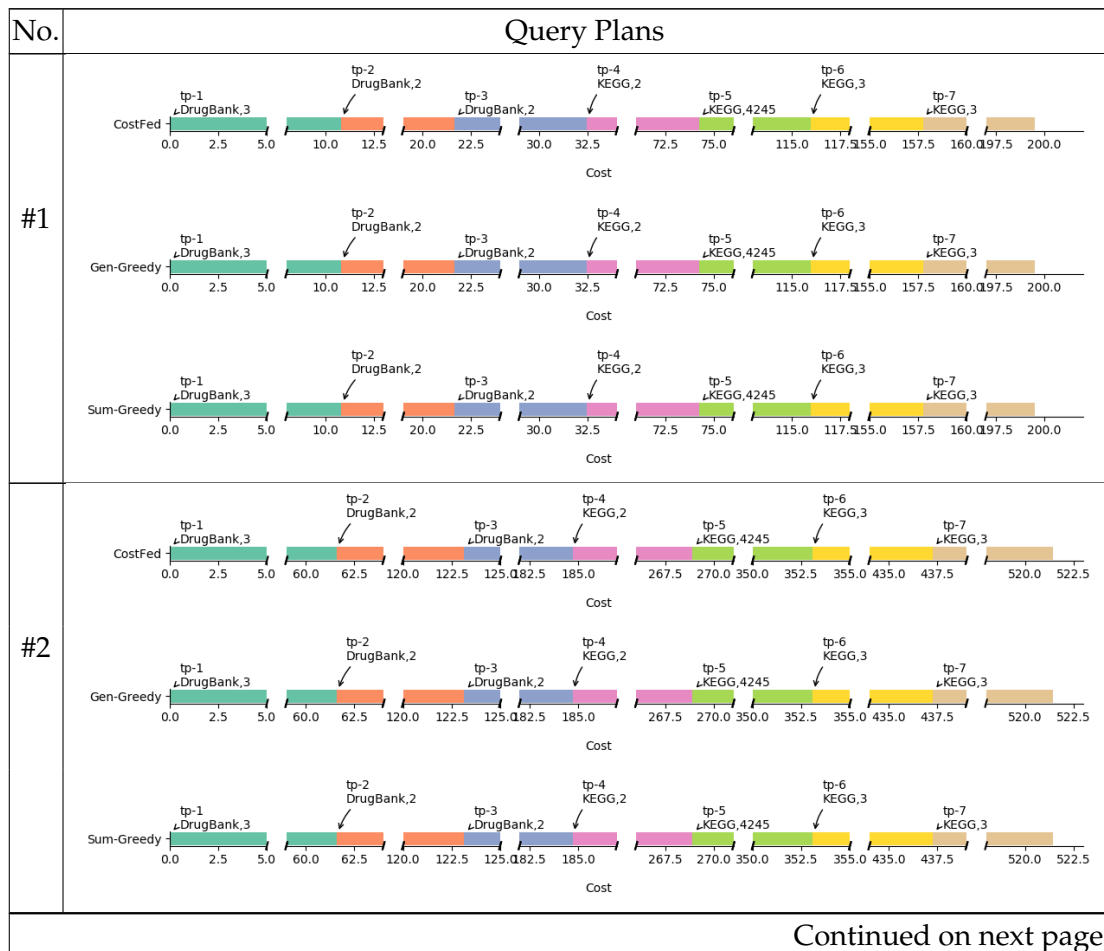


TABLE B.46: Query Plans of Query LS4 at Settings of 10 Flat Pricing Functions



Continued on next page

Table B.46 – continued from previous page

No.	Query Plans
#3	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p>
#4	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p>
#5	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p>
#6	<p>CostFed</p> <p>Gen-Greedy</p>

Continued on next page

Table B.46 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#8	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#9	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
	<p>CostFed</p> <p>Cost</p>
#10	Continued on next page

Table B.46 – continued from previous page

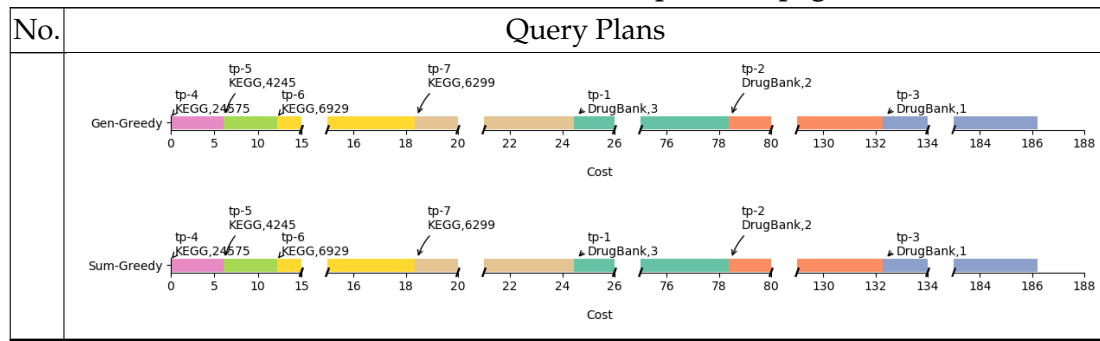
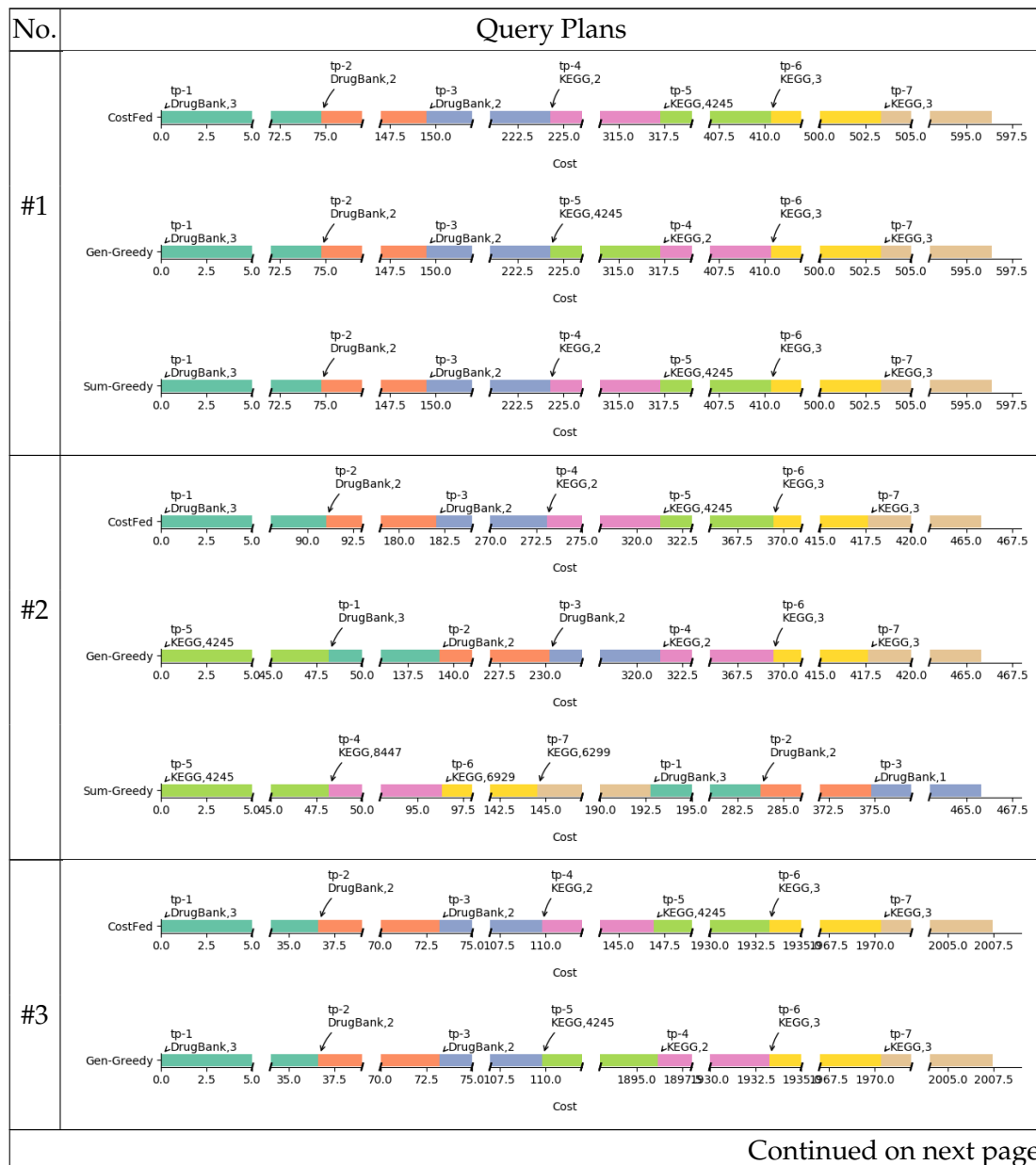


TABLE B.47: Query Plans of Query LS4 at Settings of 10 freemium Pricing Functions



Continued on next page

Table B.47 – continued from previous page

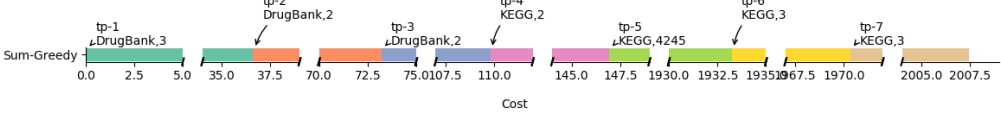
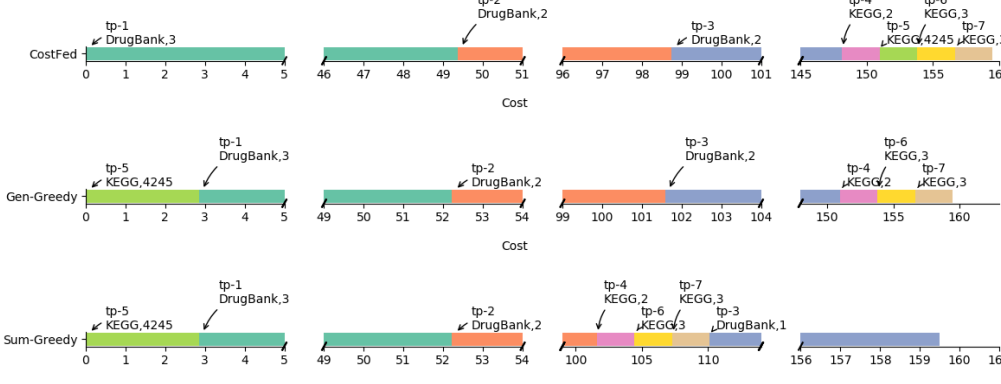
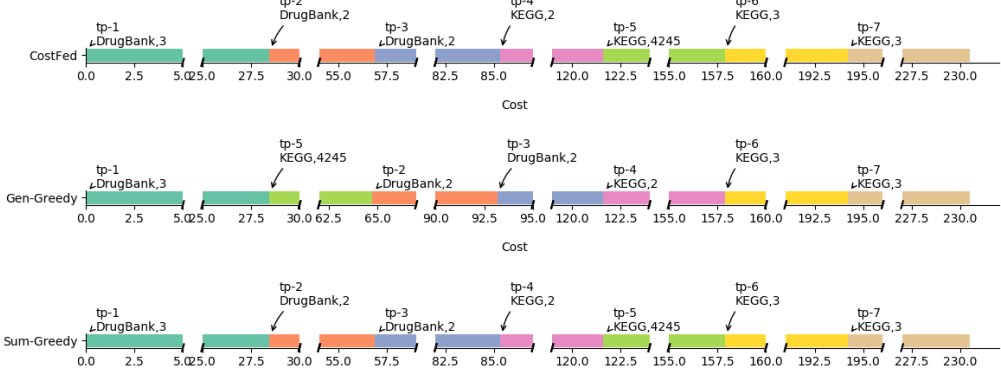
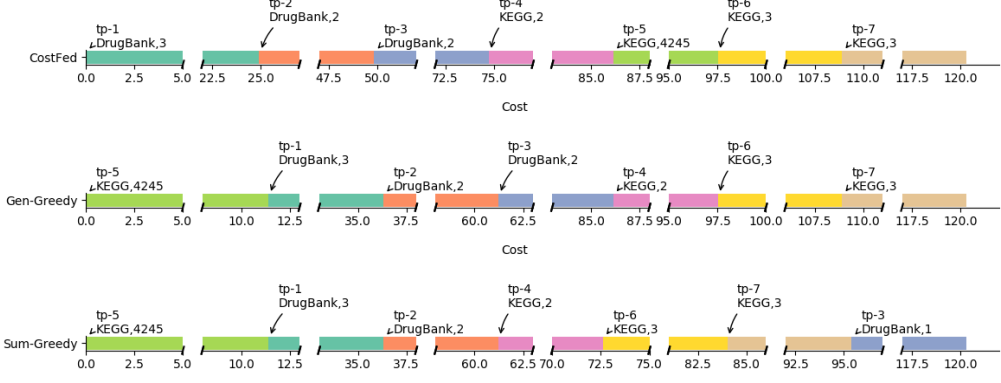
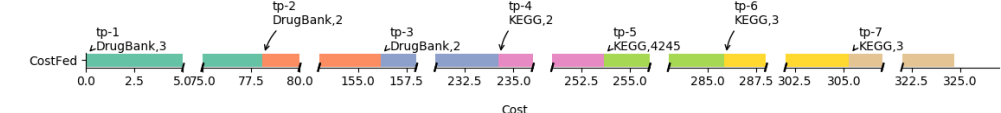
No.	Query Plans
	 <p>Sum-Greedy</p> <p>Cost</p>
#4	 <p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#5	 <p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#6	 <p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#7	 <p>CostFed</p> <p>Cost</p>
#7	Continued on next page

Table B.47 – continued from previous page

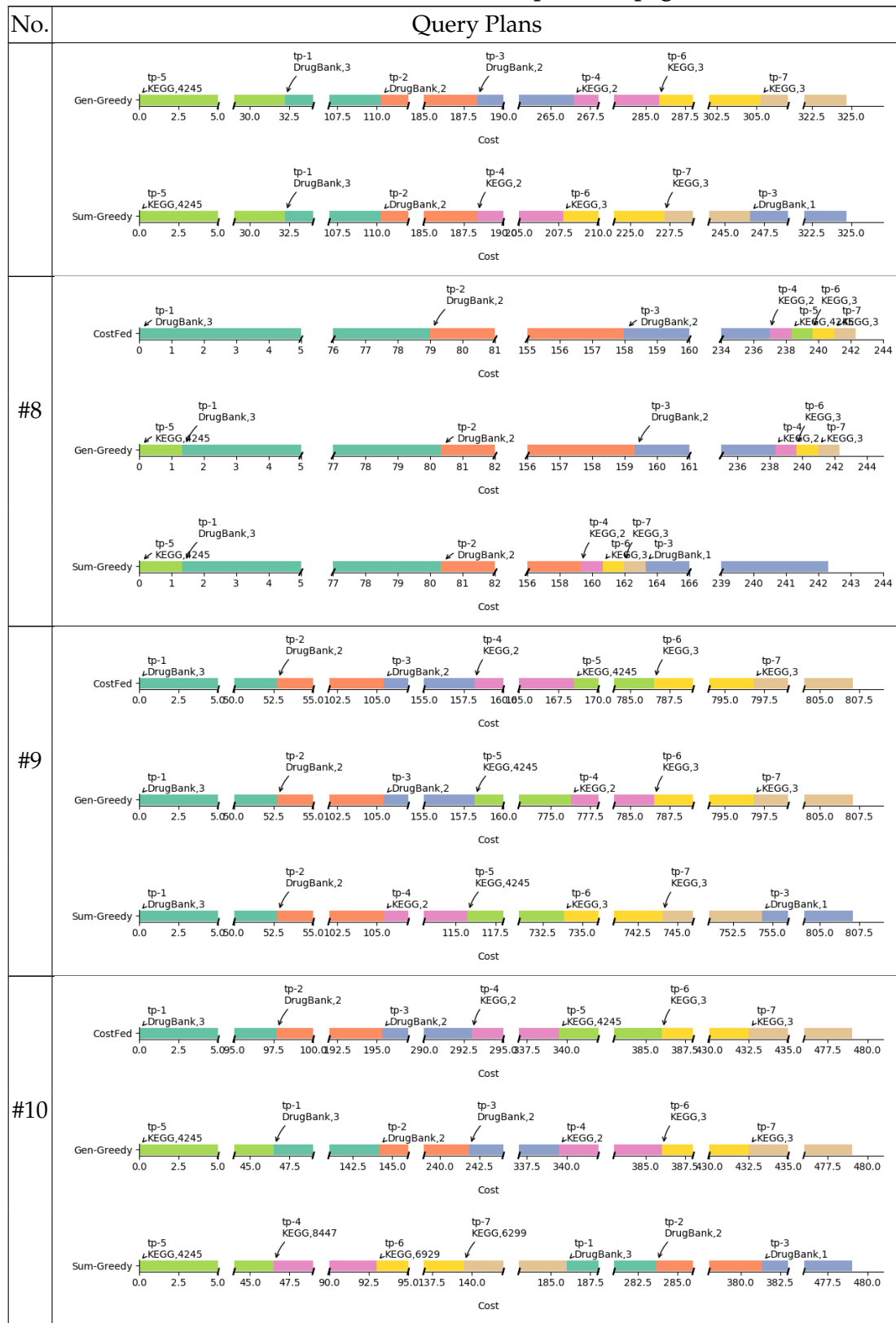


TABLE B.48: Query Plans of Query LS4 at Settings of 10 per Pricing Functions

No.	Query Plans
#1	
#2	
#3	
#4	

Continued on next page

Table B.48 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#5	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#6	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#8	<p>CostFed</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#8	Continued on next page

Table B.48 – continued from previous page

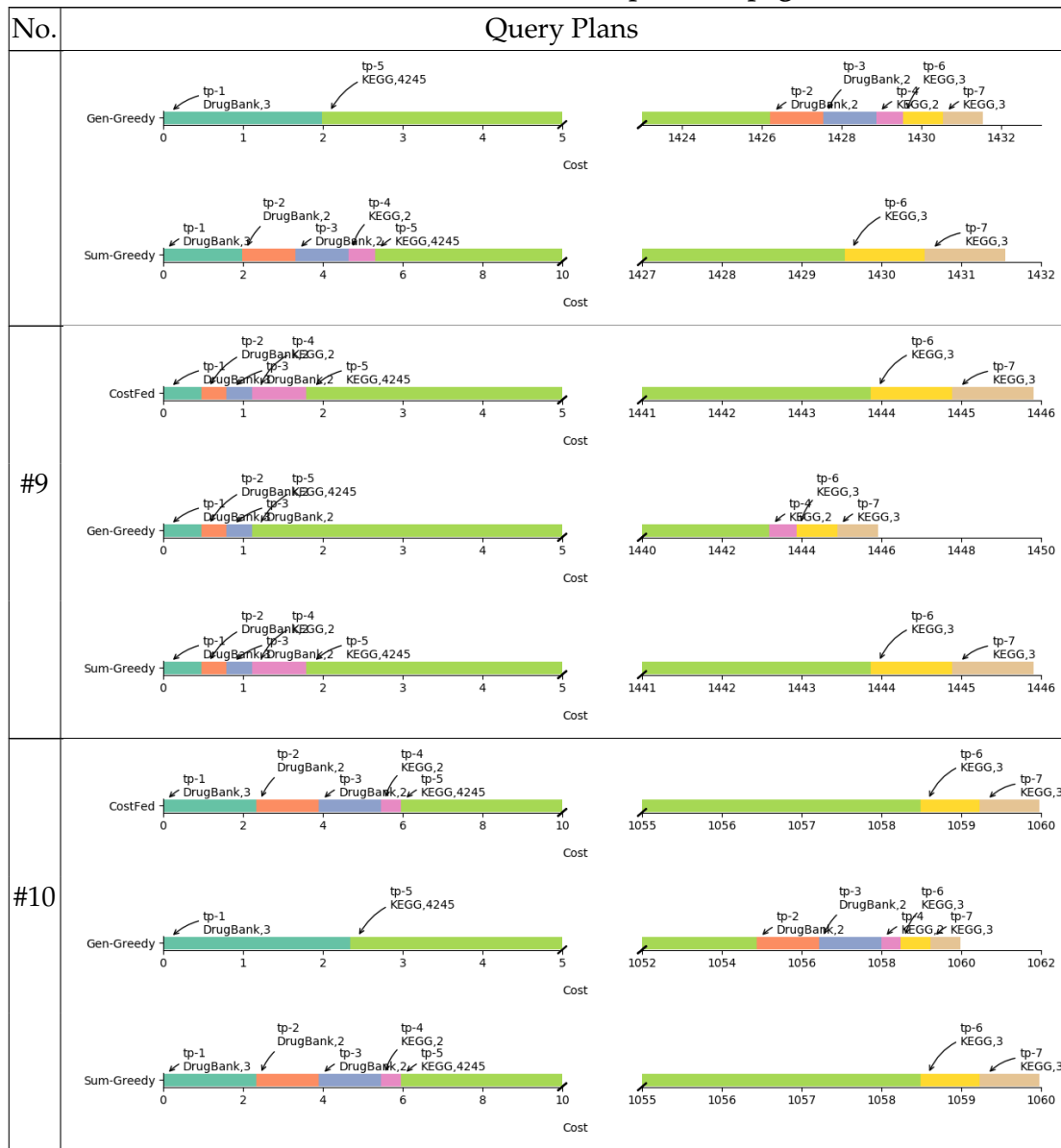
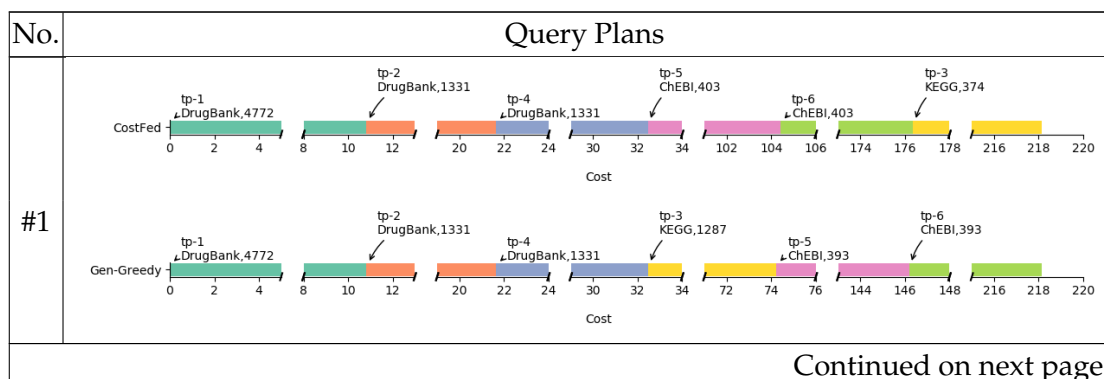


TABLE B.49: Query Plans of Query LS5 at Settings of 10 Flat Pricing Functions



Continued on next page

Table B.49 – continued from previous page

No.	Query Plans
#2	
#3	
#4	
#5	
#6	
#7	
#7	Continued on next page

Table B.49 – continued from previous page

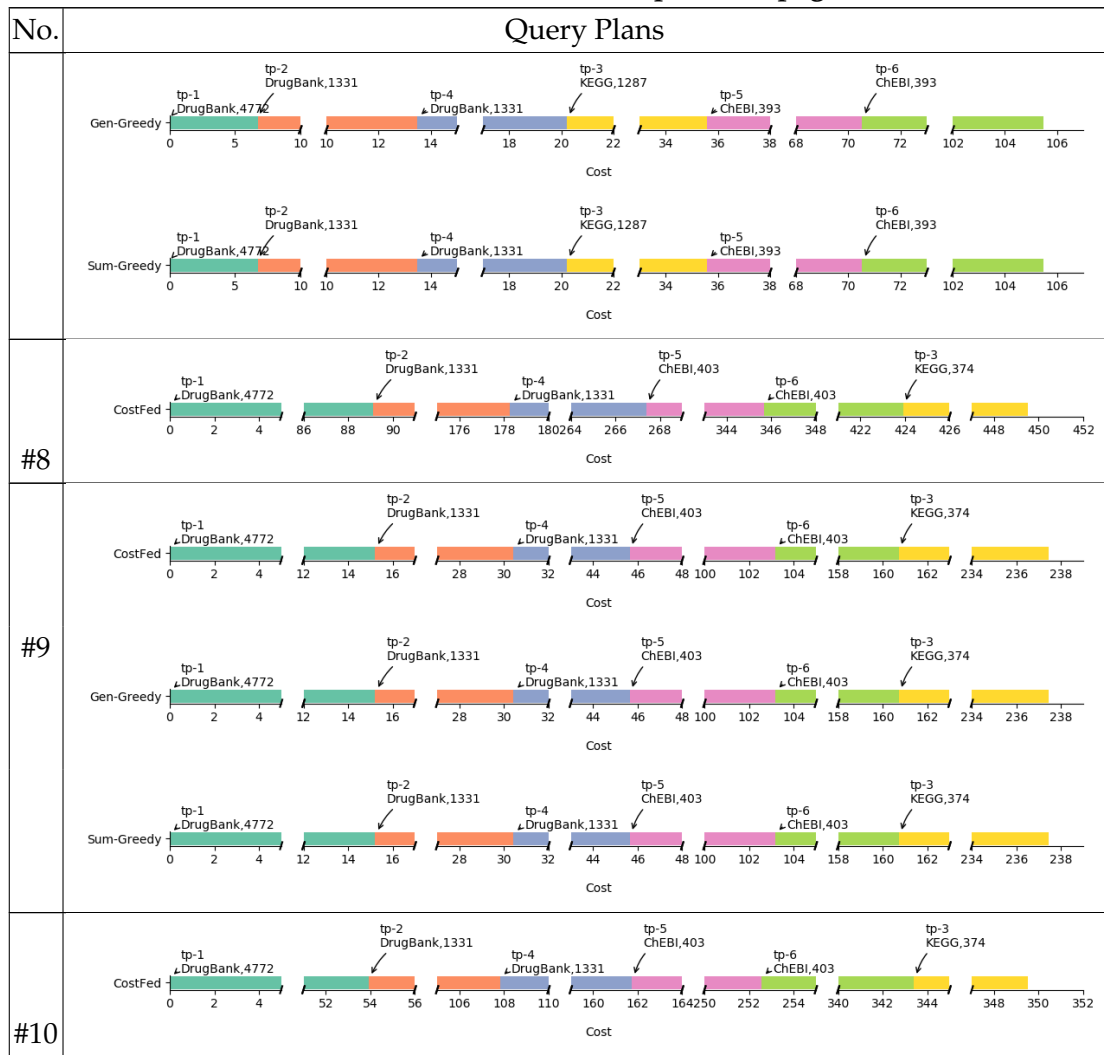
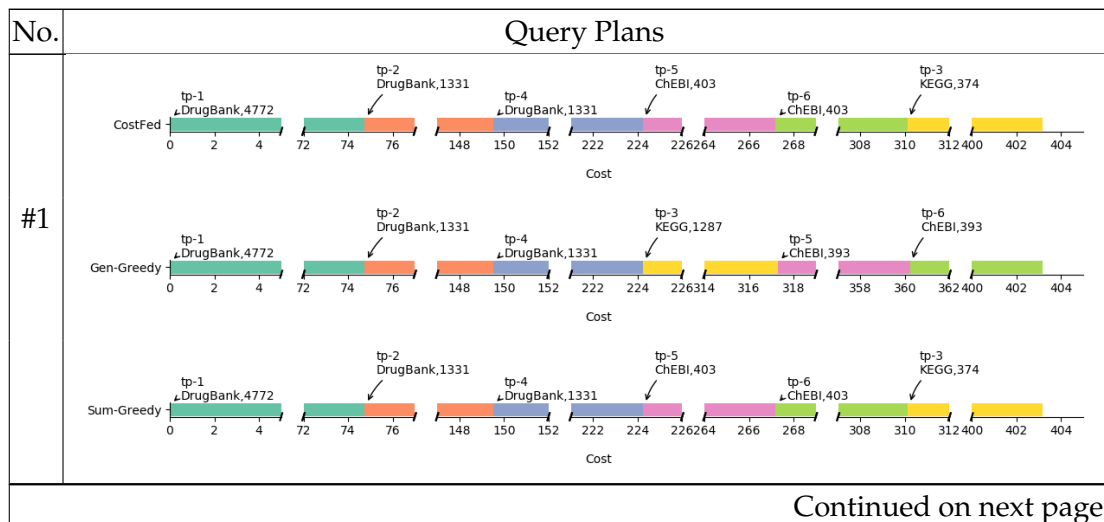


TABLE B.50: Query Plans of Query LS5 at Settings of 10 freemium Pricing Functions



Continued on next page

Table B.50 – continued from previous page

No.	Query Plans
#2	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#5	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p>
	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p>

Continued on next page

Table B.50 – continued from previous page

No.	Query Plans
#6	
#7	
#8	
#9	Continued on next page

Table B.50 – continued from previous page

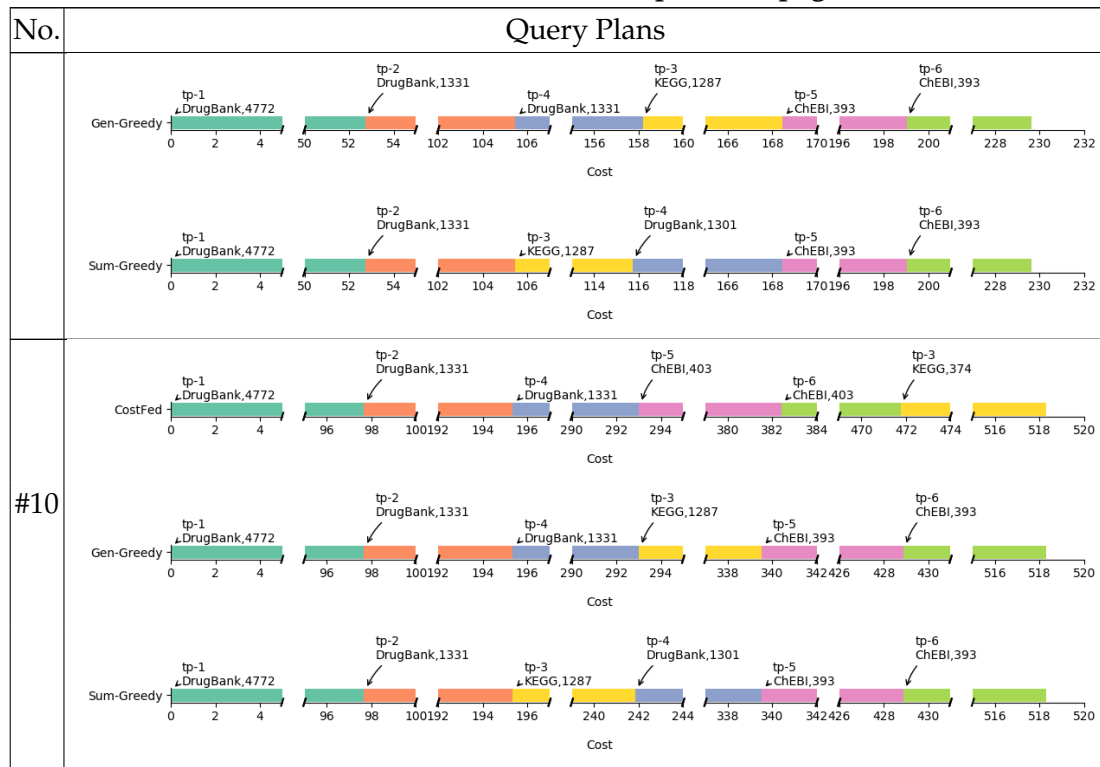
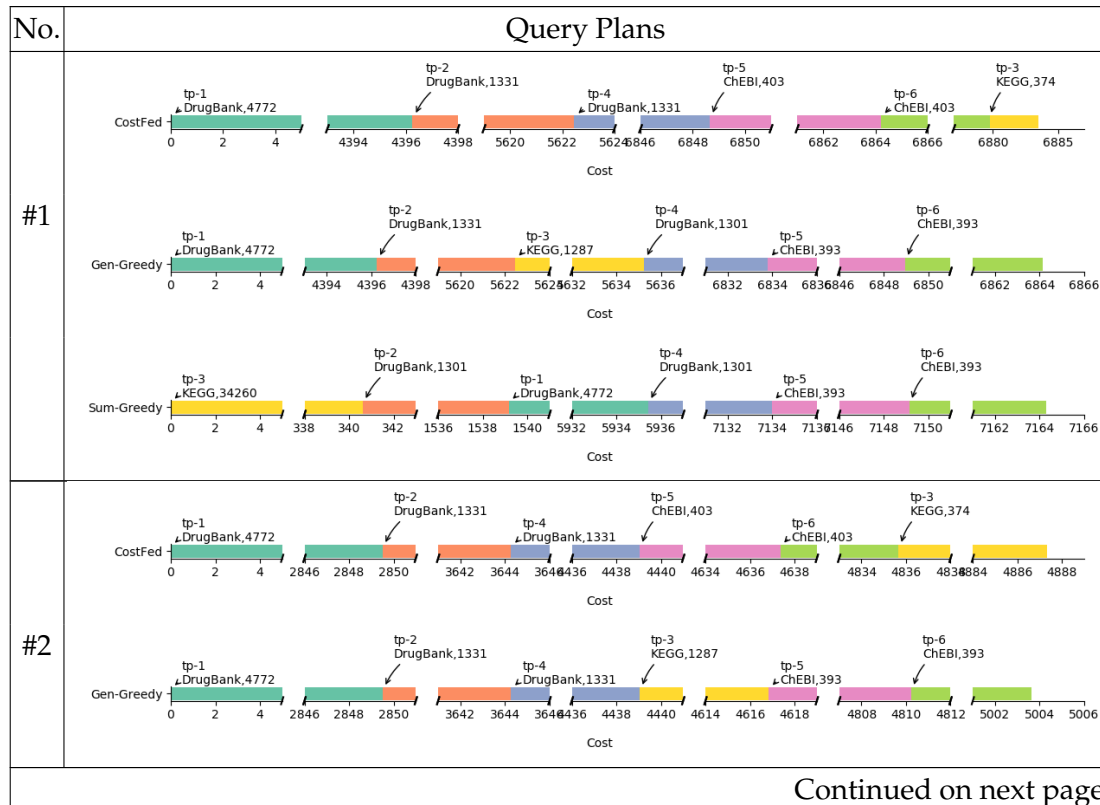


TABLE B.51: Query Plans of Query LS5 at Settings of 10 per Pricing Functions



Continued on next page

Table B.51 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#3	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#4	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#5	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#6	<p>CostFed</p> <p>Cost</p>

Continued on next page

Table B.51 – continued from previous page

No.	Query Plans
#7	<p>Gen-Greedy</p> <p>Cost</p>
	<p>Sum-Greedy</p> <p>Cost</p>
	<p>CostFed</p> <p>Cost</p>
#8	<p>Gen-Greedy</p> <p>Cost</p>
	<p>Sum-Greedy</p> <p>Cost</p>
	<p>CostFed</p> <p>Cost</p>
#9	<p>Gen-Greedy</p> <p>Cost</p>
	<p>Sum-Greedy</p> <p>Cost</p>
	<p>CostFed</p> <p>Cost</p>

Continued on next page

Table B.51 – continued from previous page

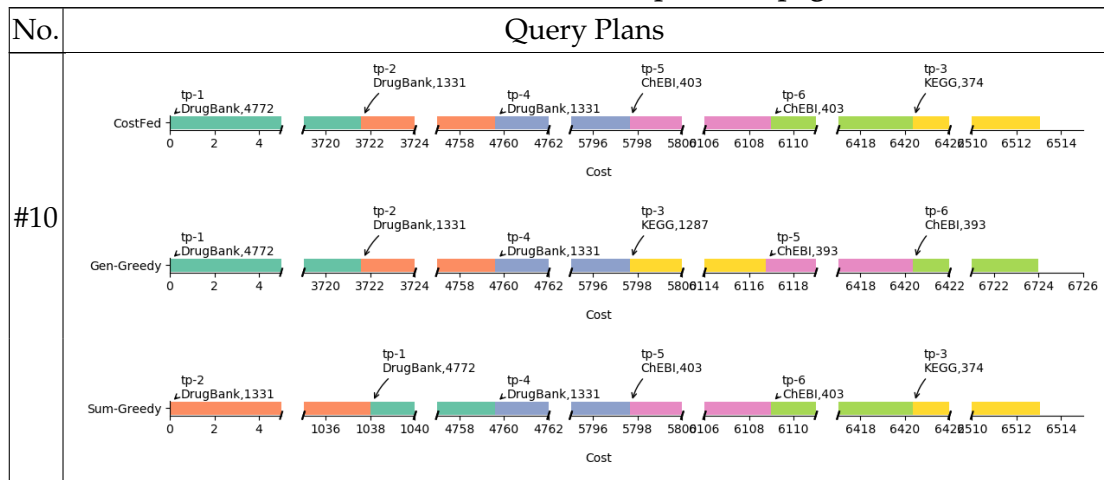
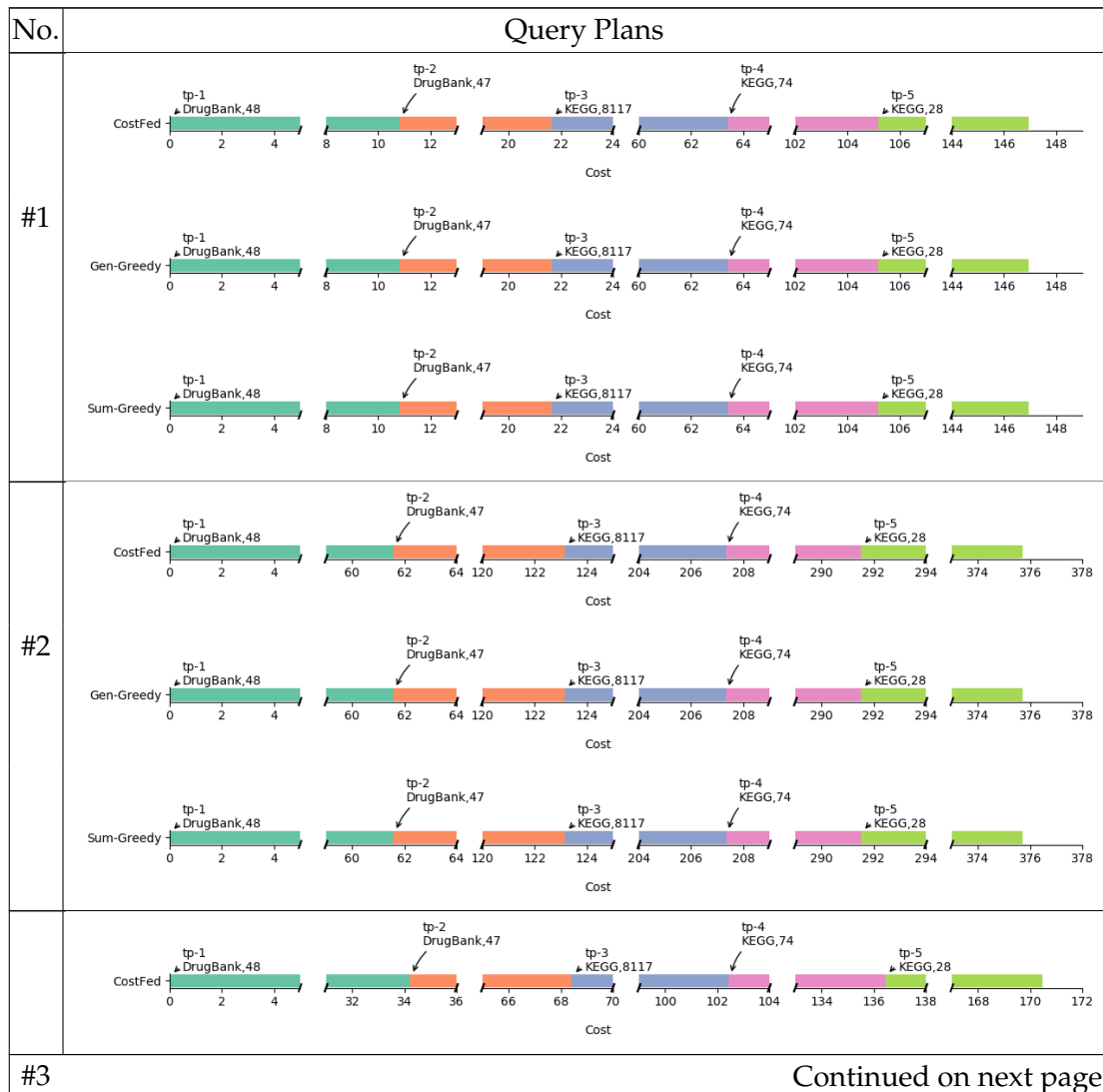
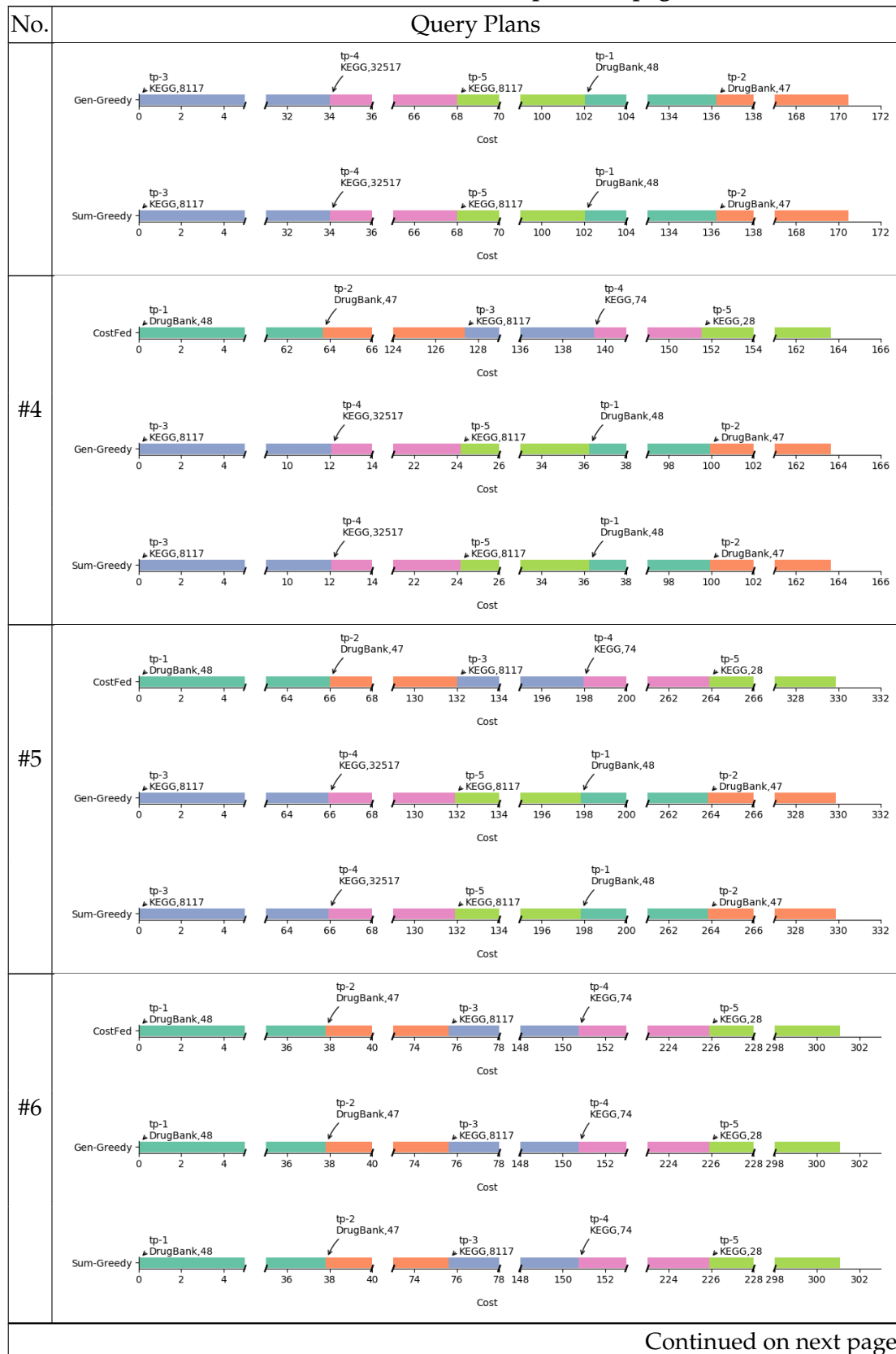


TABLE B.52: Query Plans of Query LS6 at Settings of 10 Flat Pricing Functions



Continued on next page

Table B.52 – continued from previous page



Continued on next page

Table B.52 – continued from previous page

No.	Query Plans
#7	<p>Query plan #7 shows three horizontal bar charts for CostFed, Gen-Greedy, and Sum-Greedy. Each chart displays five segments representing different join orders. The x-axis is labeled 'Cost' with values from 0.0 to 62. The segments are labeled: tp-1 DrugBank,48 (green), tp-2 DrugBank,47 (orange), tp-3 KEGG,8117 (blue), tp-4 KEGG,74 (purple), and tp-5 KEGG,28 (pink).</p>
#8	<p>Query plan #8 shows three horizontal bar charts for CostFed, Gen-Greedy, and Sum-Greedy. Each chart displays five segments representing different join orders. The x-axis is labeled 'Cost' with values from 0 to 256. The segments are labeled: tp-1 DrugBank,48 (green), tp-2 DrugBank,47 (orange), tp-3 KEGG,8117 (blue), tp-4 KEGG,32517 (purple), and tp-5 KEGG,8117 (pink).</p>
#9	<p>Query plan #9 shows three horizontal bar charts for CostFed, Gen-Greedy, and Sum-Greedy. Each chart displays five segments representing different join orders. The x-axis is labeled 'Cost' with values from 0 to 262. The segments are labeled: tp-1 DrugBank,48 (green), tp-2 DrugBank,47 (orange), tp-3 KEGG,8117 (blue), tp-4 KEGG,74 (purple), and tp-5 KEGG,28 (pink).</p>
#10	<p>Query plan #10 shows two horizontal bar charts for CostFed and Gen-Greedy. Each chart displays five segments representing different join orders. The x-axis is labeled 'Cost' with values from 0 to 128. The segments are labeled: tp-1 DrugBank,48 (green), tp-2 DrugBank,47 (orange), tp-3 KEGG,8117 (blue), tp-4 KEGG,32517 (purple), and tp-5 KEGG,8117 (pink).</p>

Continued on next page

Table B.52 – continued from previous page

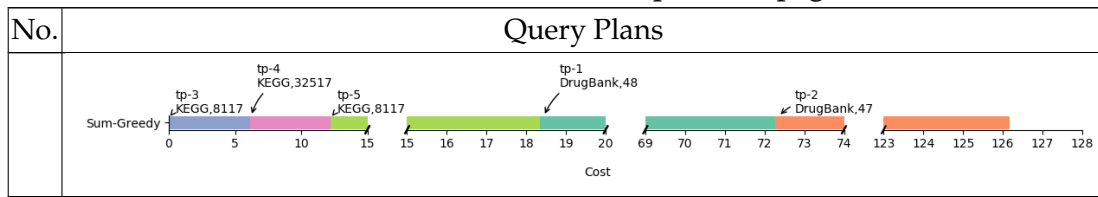
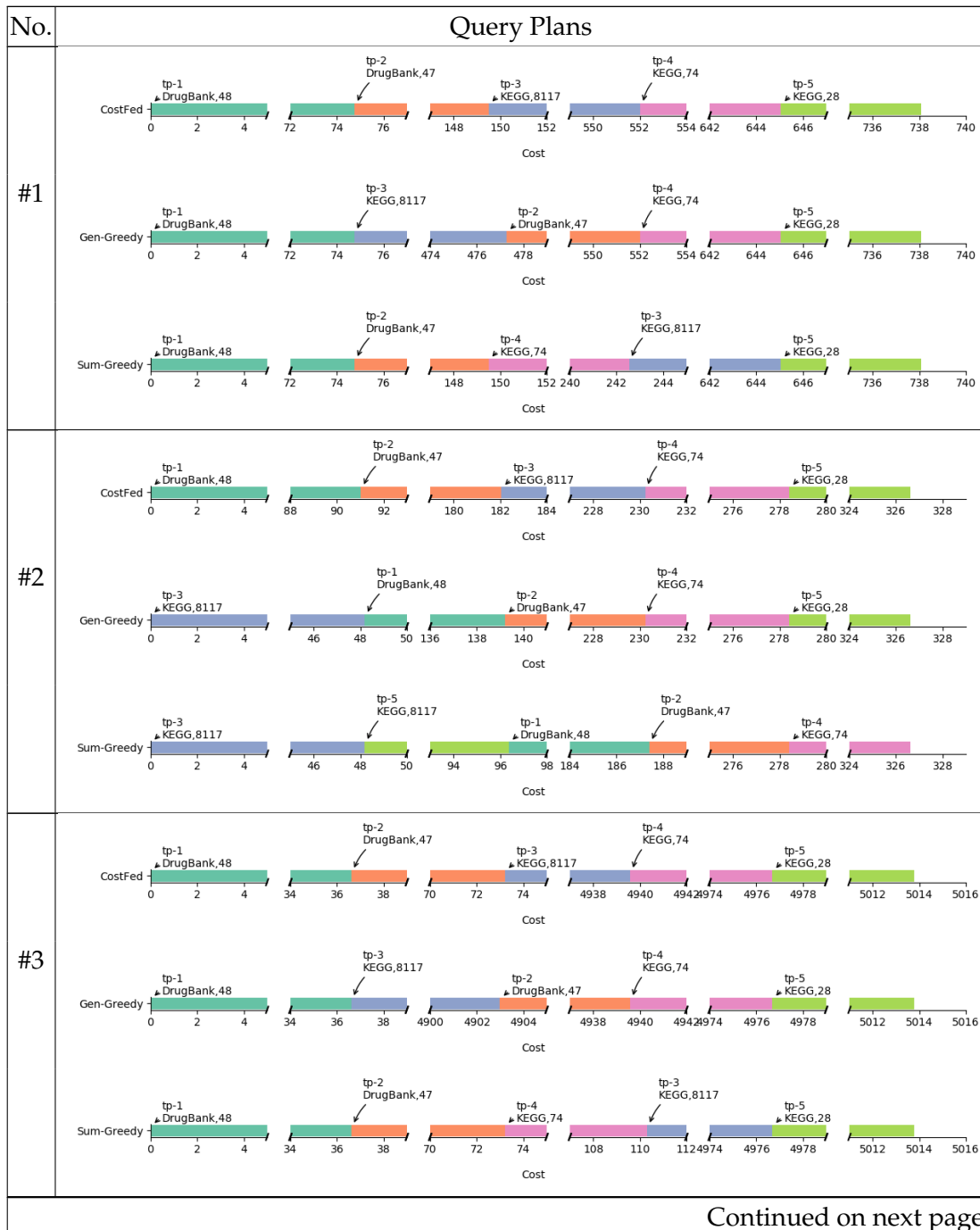


TABLE B.53: Query Plans of Query LS6 at Settings of 10 freemium Pricing Functions



Continued on next page

Table B.53 – continued from previous page

No.	Query Plans
#4	<p>Query plan #4 shows three execution strategies: CostFed, Gen-Greedy, and Sum-Greedy. Each strategy involves a sequence of five operations: tp-1 DrugBank,48 (green), tp-2 DrugBank,47 (orange), tp-3 KEGG,8117 (blue), tp-4 KEGG,74 (pink), and tp-5 KEGG,28 (light green). The x-axis represents the cumulative cost, with values ranging from 0 to 670. The CostFed plan has a total cost of 670, Gen-Greedy has 670, and Sum-Greedy has 672.</p>
#5	<p>Query plan #5 shows three execution strategies: CostFed, Gen-Greedy, and Sum-Greedy. Each strategy involves a sequence of five operations: tp-1 DrugBank,48 (green), tp-2 DrugBank,47 (orange), tp-3 KEGG,8117 (blue), tp-4 KEGG,74 (pink), and tp-5 KEGG,28 (light green). The x-axis represents the cumulative cost, with values ranging from 0 to 1726. The CostFed plan has a total cost of 1726, Gen-Greedy has 1726, and Sum-Greedy has 1726.</p>
#6	<p>Query plan #6 shows three execution strategies: CostFed, Gen-Greedy, and Sum-Greedy. Each strategy involves a sequence of five operations: tp-1 DrugBank,48 (green), tp-2 DrugBank,47 (orange), tp-3 KEGG,8117 (blue), tp-4 KEGG,74 (pink), and tp-5 KEGG,28 (light green). The x-axis represents the cumulative cost, with values ranging from 0 to 1146. The CostFed plan has a total cost of 1146, Gen-Greedy has 1146, and Sum-Greedy has 1146.</p>
#7	<p>Query plan #7 shows two execution strategies: CostFed and Gen-Greedy. Each strategy involves a sequence of five operations: tp-1 DrugBank,48 (green), tp-2 DrugBank,47 (orange), tp-3 KEGG,8117 (blue), tp-4 KEGG,74 (pink), and tp-5 KEGG,28 (light green). The x-axis represents the cumulative cost, with values ranging from 0 to 688. The CostFed plan has a total cost of 688, and Gen-Greedy has 688.</p>

Continued on next page

Table B.53 – continued from previous page

No.	Query Plans
#8	<p>Sum-Greedy</p> <p>Cost</p>
	<p>CostFed</p> <p>Cost</p>
	<p>Gen-Greedy</p> <p>Cost</p>
	<p>Sum-Greedy</p> <p>Cost</p>
#9	<p>CostFed</p> <p>Cost</p>
	<p>Gen-Greedy</p> <p>Cost</p>
	<p>Sum-Greedy</p> <p>Cost</p>
#10	<p>CostFed</p> <p>Cost</p>
	<p>Gen-Greedy</p> <p>Cost</p>
	<p>Sum-Greedy</p> <p>Cost</p>

TABLE B.54: Query Plans of Query LS6 at Settings of 10 per Pricing Functions

No.	Query Plans
#1	<p>Query plan #1 shows three execution plans: CostFed, Gen-Greedy, and Sum-Greedy. Each plan consists of five operations: tp-1 DrugBank,48 (green), tp-2 DrugBank,47 (orange), tp-3 KEGG,8117 (blue), tp-4 KEGG,74 (purple), and tp-5 KEGG,2 (green). The x-axis represents cost, with segments labeled by operation and total cost values.</p>
#2	<p>Query plan #2 shows three execution plans: CostFed, Gen-Greedy, and Sum-Greedy. Each plan consists of five operations: tp-1 DrugBank,48 (green), tp-2 DrugBank,47 (orange), tp-3 KEGG,8117 (blue), tp-4 KEGG,74 (purple), and tp-5 KEGG,28 (green). The x-axis represents cost, with segments labeled by operation and total cost values.</p>
#3	<p>Query plan #3 shows three execution plans: CostFed, Gen-Greedy, and Sum-Greedy. Each plan consists of five operations: tp-1 DrugBank,48 (green), tp-2 DrugBank,47 (orange), tp-3 KEGG,8117 (blue), tp-4 KEGG,74 (purple), and tp-5 KEGG,28 (green). The x-axis represents cost, with segments labeled by operation and total cost values.</p>
#4	<p>Query plan #4 shows two execution plans: CostFed and Gen-Greedy. Each plan consists of five operations: tp-1 DrugBank,48 (green), tp-2 DrugBank,47 (orange), tp-3 KEGG,8117 (blue), tp-4 KEGG,74 (purple), and tp-5 KEGG,28 (green). The x-axis represents cost, with segments labeled by operation and total cost values.</p>

Continued on next page

Table B.54 – continued from previous page

No.	Query Plans
	<p>Sum-Greedy</p> <p>Cost</p>
#5	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#6	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#7	<p>CostFed</p> <p>Cost</p> <p>Gen-Greedy</p> <p>Cost</p> <p>Sum-Greedy</p> <p>Cost</p>
#8	<p>CostFed</p> <p>Cost</p>

Continued on next page

Table B.54 – continued from previous page

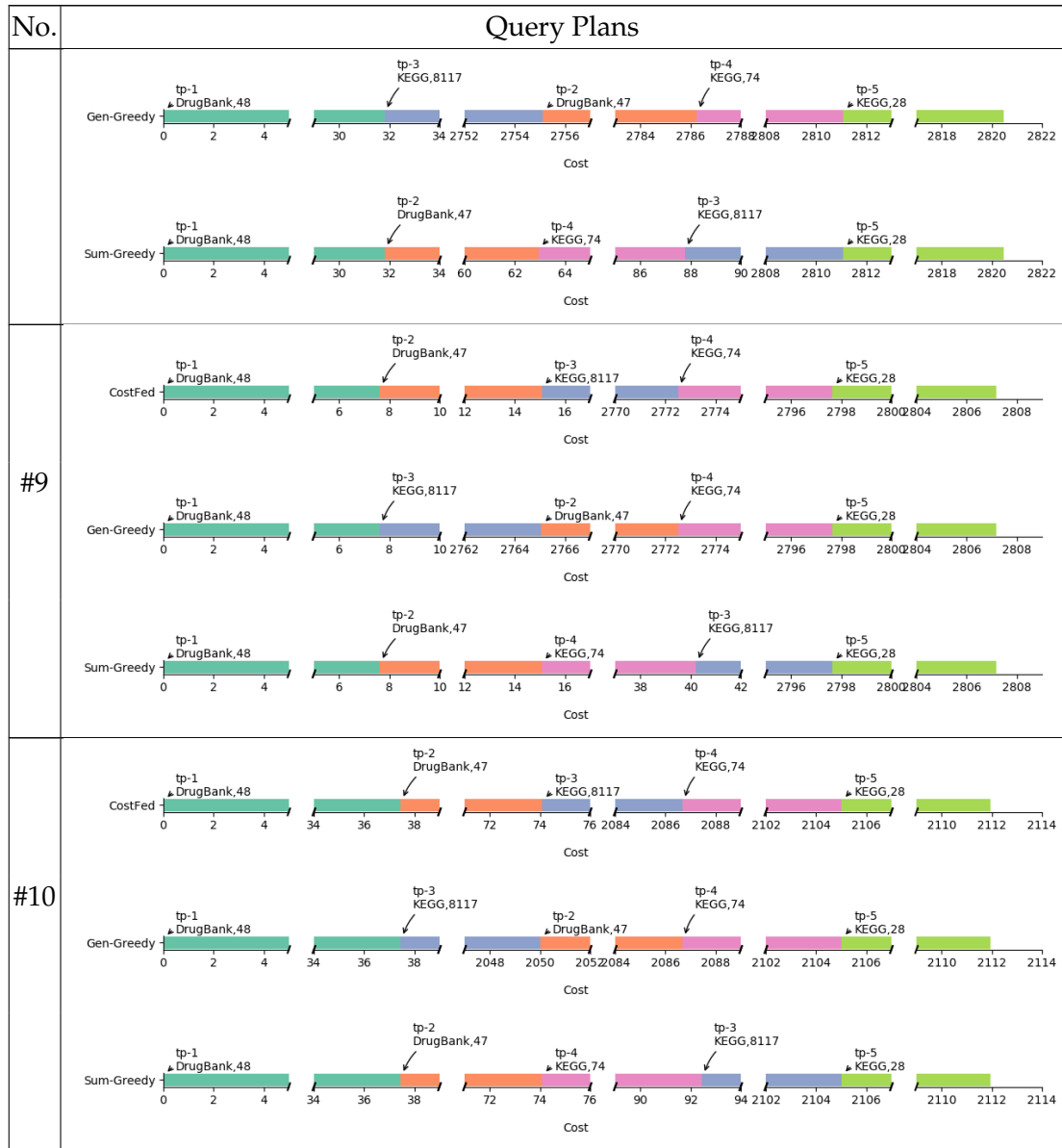
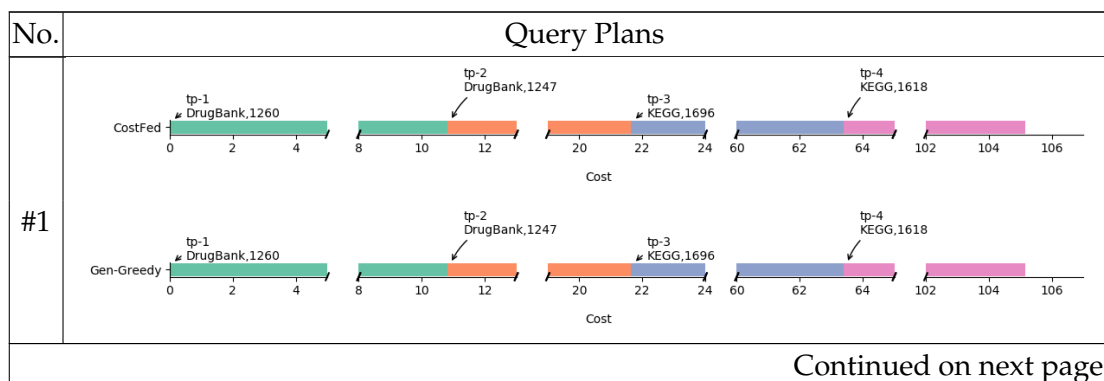


TABLE B.55: Query Plans of Query LS7 at Settings of 10 Flat Pricing Functions



Continued on next page

Table B.55 – continued from previous page

No.	Query Plans
#2	
#3	
#4	
#5	
#6	
#7	
Continued on next page	

Table B.55 – continued from previous page

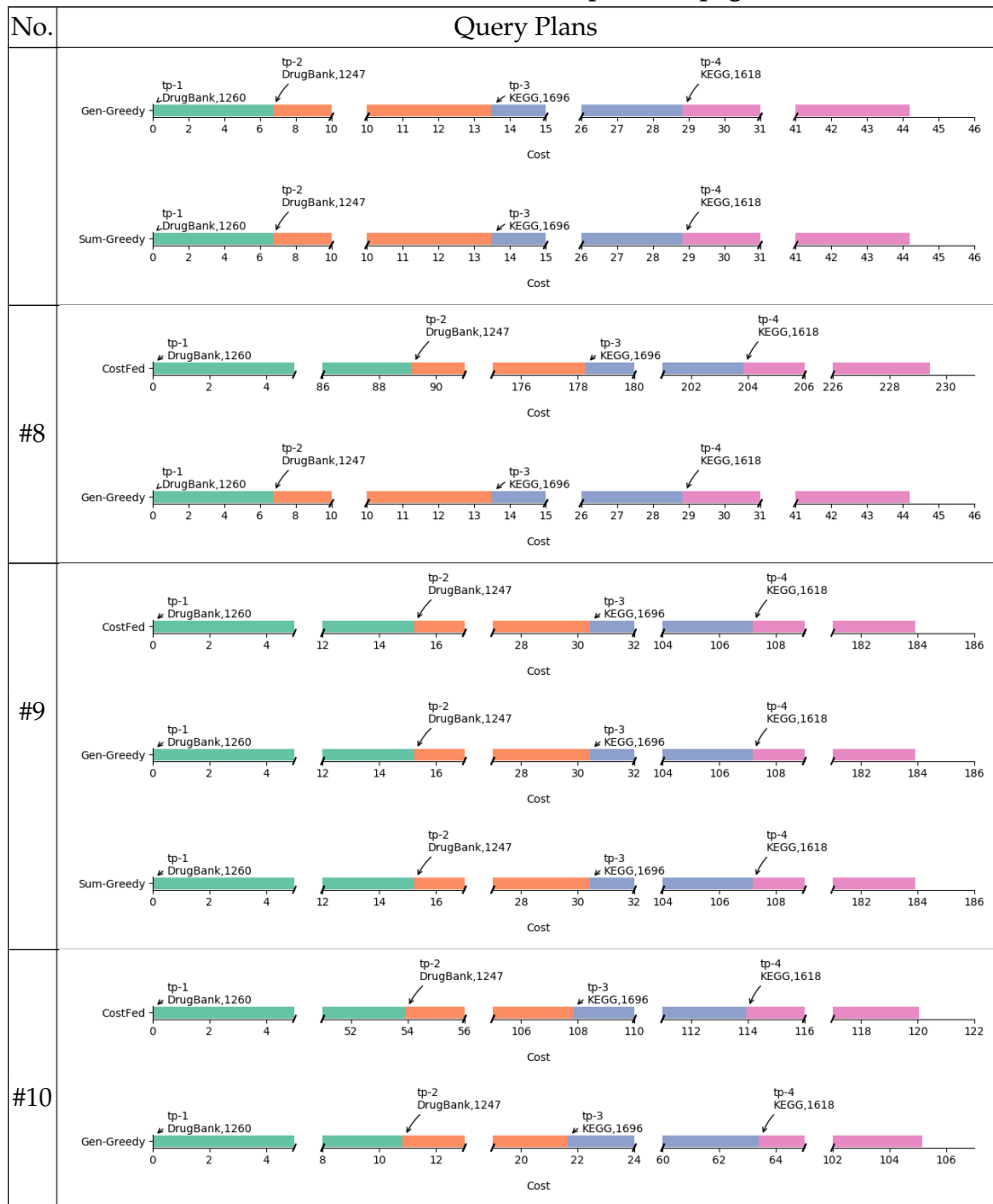


TABLE B.56: Query Plans of Query LS7 at Settings of 10 freemium Pricing Functions

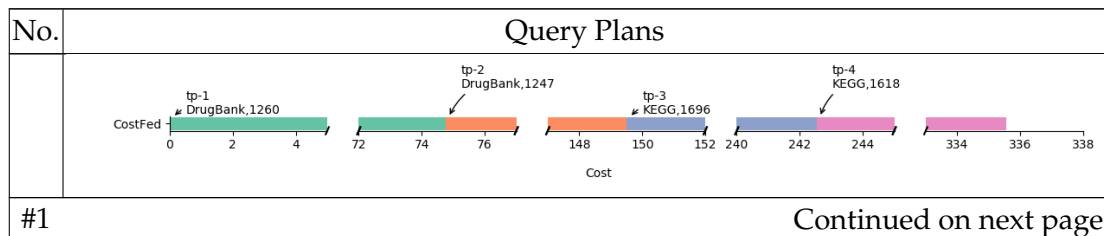


Table B.56 – continued from previous page

No.	Query Plans
#2	
#3	
#4	
Continued on next page	

Table B.56 – continued from previous page

No.	Query Plans
#5	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p>
#6	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p>
#7	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p>
#8	<p>CostFed</p> <p>Gen-Greedy</p>

Continued on next page

Table B.56 – continued from previous page

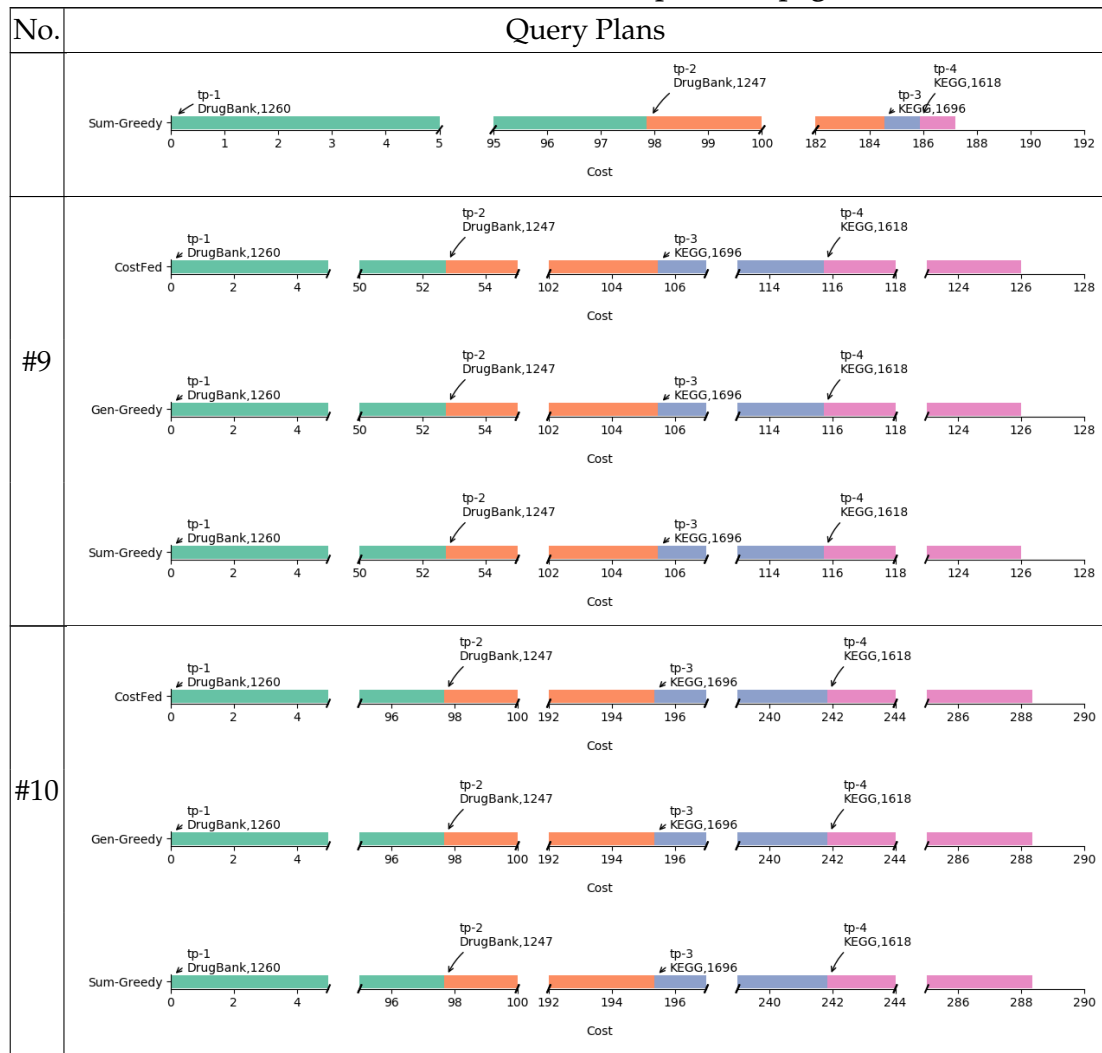
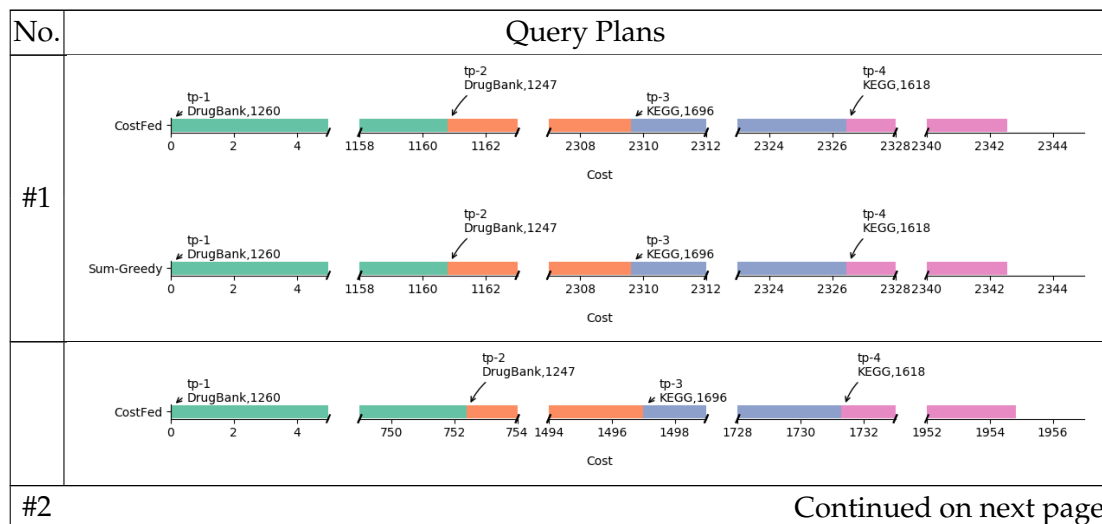


TABLE B.57: Query Plans of Query LS7 at Settings of 10 per Pricing Functions



Continued on next page

Table B.57 – continued from previous page

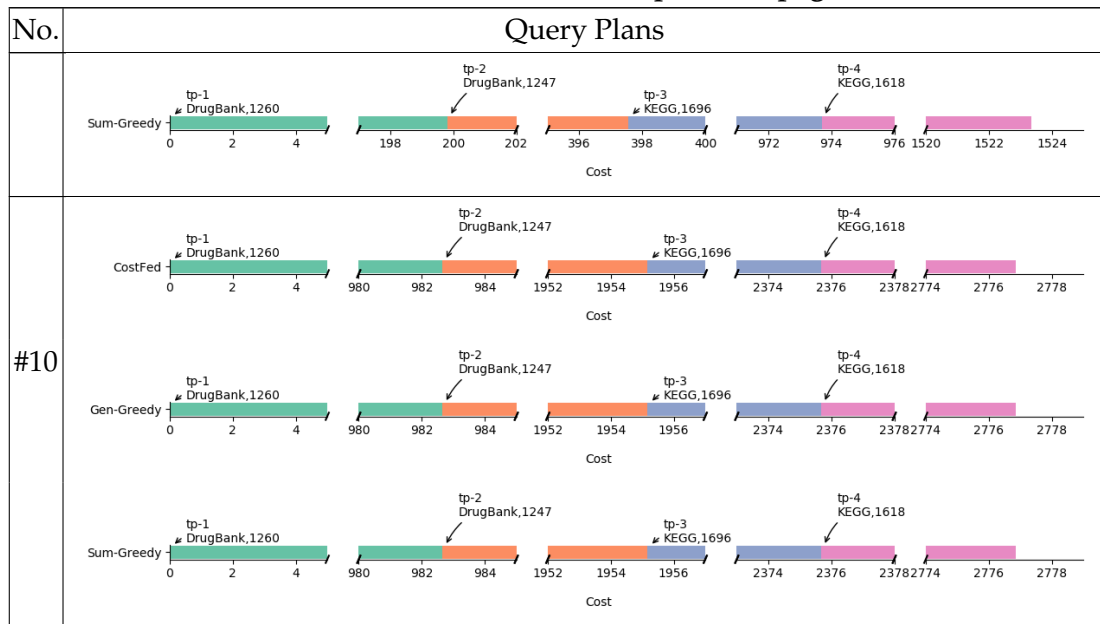
No.	Query Plans
	<p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#3	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#4	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
#5	<p>CostFed</p> <p>Gen-Greedy</p> <p>Sum-Greedy</p> <p>Cost</p>
Continued on next page	

Table B.57 – continued from previous page

No.	Query Plans
#6	
#7	
#8	
#9	

Continued on next page

Table B.57 – continued from previous page



B.3 Entire Experiment Results

B.3.1 Cost Efficiency

B.4 Entire Experiment Results

B.4.1 Cost

TABLE B.58: Experiment Results of Query Answer Cost with Settings of Flat

Query	Alg	Flat									
		1	2	3	4	5	6	7	8	9	10
CD1	G	187.98	4.95	47.67	177.48	2.87	193.32	174.81	198.74	165.5	155.71
	P	187.98	X	47.67	177.48	2.87	193.32	X	198.74	X	155.71
	C	187.98	4.95	47.67	177.48	2.87	193.32	174.81	198.74	165.5	155.71
CD2	G	255.96	26.11	69.54	271.62	5.79	235.8	206.65	269.26	263.3	245.96
	P	255.96	26.11	69.54	271.62	5.79	235.8	206.65	269.26	263.3	245.96
	C	255.96	26.11	69.54	271.62	5.79	235.8	206.65	269.26	263.3	245.96
CD3	G	391.93	68.42	113.28	459.89	11.63	320.78	270.34	410.31	458.91	426.48
	P	391.93	68.42	113.28	459.89	11.63	320.78	270.34	410.31	458.91	426.48
	C	391.93	68.42	113.28	459.89	11.63	320.78	270.34	410.31	458.91	426.48
CD4	G	471.83	260.95	265.03	423.61	17.43	456.07	305.45	283.31	356.22	383.18
	P	471.83	260.95	265.03	423.61	17.43	456.07	305.45	283.31	356.22	383.18

Continued on next page

Table B.58 – continued from previous page

Query	Alg	Flat									
		1	2	3	4	5	6	7	8	9	10
CD5	C	471.83	260.95	265.03	423.61	17.43	456.07	305.45	283.31	356.22	383.18
	G	325.21	212.98	188.65	352.36	15.55	260.14	150.78	197.43	322.75	332.17
	P	325.21	212.98	188.65	352.36	15.55	260.14	150.78	197.43	322.75	332.17
	C	325.21	212.98	188.65	352.36	15.55	260.14	150.78	197.43	322.75	332.17
CD6	G	270.8	300.52	264.72	262.2	309.09	258.62	276.83	342.53	287.78	252.73
	P	372.93	302.99	288.56	380.16	310.52	420.18	364.23	508.66	423.21	342.37
	C	262.66	243.7	186.98	232.99	235.61	193.73	211.28	275.77	235.1	240.94
CD7	G	563.91	145.61	106.32	483.75	42.13	504.52	384.25	587.54	425.53	387.33
	P	563.91	145.61	106.32	483.75	42.13	504.52	384.25	587.54	425.53	387.33
	C	563.91	145.61	106.32	483.75	42.13	504.52	384.25	587.54	425.53	387.33
LD6	G	419.19	280.83	194.14	416.75	33.74	319.08	168.1	292.46	370.01	370.13
	P	419.19	280.83	X	X	33.74	319.08	X	292.46	X	X
	C	419.19	280.83	194.14	416.75	33.74	319.08	168.1	292.46	370.01	370.13
LD7	G	187.96	135.71	10.98	128.79	36.4	117.88	34.64	190.06	94.53	75.92
	P	187.96	135.71	10.98	128.79	36.4	117.88	34.64	190.06	94.53	75.92
	C	187.96	135.71	10.98	128.79	36.4	117.88	34.64	190.06	94.53	75.92
LD8	G	168.44	227.04	146.37	379.35	203.88	198.38	83.89	408.45	241.24	342.26
	P	168.44	X	X	379.35	X	198.38	83.89	X	241.24	342.26
	C	168.44	227.04	146.37	379.35	203.88	198.38	83.89	408.45	241.24	342.26
LD10	G	229.96	44.79	67.58	277.01	7.27	181.63	151.09	240.42	278.35	258.37
	P	229.96	44.79	67.58	277.01	7.27	181.63	151.09	240.42	278.35	258.37
	C	229.96	44.79	67.58	277.01	7.27	181.63	151.09	240.42	278.35	258.37
LD11	G	339.93	105.79	109.35	470.69	14.6	212.43	159.21	352.62	489.01	451.3
	P	339.93	105.79	109.35	470.69	14.6	212.43	159.21	352.62	489.01	451.3
	C	339.93	105.79	109.35	470.69	14.6	212.43	159.21	352.62	489.01	451.3
LS2	G	78.81	82.73	56.08	157.83	68.93	80.29	38.58	159.66	113.02	144.17
	P	78.81	82.73	56.08	157.83	68.93	80.29	38.58	159.66	113.02	144.17
	C	78.81	82.73	56.08	157.83	68.93	80.29	38.58	159.66	113.02	144.17
LS3	G	111.27	267.45	158.7	348.9	266.97	193.7	58.78	427.07	158.66	305.92
	P	111.27	267.45	158.7	348.9	266.97	193.7	58.78	427.07	158.66	305.92
	C	111.27	267.45	158.7	348.9	266.97	193.7	58.78	427.07	158.66	305.92
LS4	G	199.52	521.45	238.68	239.42	461.84	414.02	81.65	369.75	352.59	186.2
	P	199.52	521.45	238.68	239.42	461.84	414.02	81.65	369.75	352.59	186.2
	C	199.52	521.45	238.68	239.42	461.84	414.02	81.65	369.75	352.59	186.2
LS5	G	218.13	416.19	238.36	354.19	423.66	365.6	105.49	449.51	237.46	349.51
	P	218.13	416.19	X	X	X	365.6	105.49	X	237.46	X
	C	218.13	416.19	238.36	354.19	423.66	365.6	105.49	449.51	237.46	349.51

Continued on next page

Table B.58 – continued from previous page

Query	Alg	Flat									
		1	2	3	4	5	6	7	8	9	10
LS6	G	146.93	375.7	170.46	163.64	329.87	301.06	59.56	255.03	260.64	126.17
	P	146.93	375.7	170.46	163.64	329.87	301.06	59.56	255.03	260.64	126.17
	C	146.93	375.7	170.46	163.64	329.87	301.06	59.56	255.03	260.64	126.17
LS7	G	105.17	291.51	136.45	151.56	263.92	225.91	44.19	229.44	183.9	120.06
	P	105.17	291.51	X	X	X	225.91	44.19	X	183.9	X
	C	105.17	291.51	136.45	151.56	263.92	225.91	44.19	229.44	183.9	120.06

TABLE B.59: Experiment Results of Query Answer Cost with Settings of Freemium

Query	Alg	Freemium									
		1	2	3	4	5	6	7	8	9	10
CD1	G	175.58	22.81	164.34	174.66	162.44	130.67	77.32	150.61	148.06	169.08
	P	175.58	22.81	164.34	174.66	162.44	130.67	77.32	150.61	148.06	169.08
	C	413.41	377.8	492.98	536.15	283.95	272.19	354.02	433.04	392.85	578.58
CD2	G	184.93	40.94	176.88	178.42	240.3	159.05	118.54	174.37	183.85	207.18
	P	184.93	40.94	176.88	178.42	240.3	159.05	118.54	174.37	183.85	207.18
	C	7199.46	2614.51	1358.22	293.85	240.3	159.05	1838.67	174.37	1105.98	2720.5
CD3	G	1020.4	77.21	201.95	10234.23	1649.93	215.83	200.98	434.25	2400.22	283.39
	P	1020.4	77.21	201.95	10234.23	1649.93	215.83	200.98	434.25	2400.22	283.39
	C	8034.93	2650.77	1383.29	10349.65	1649.93	215.83	1921.11	434.25	3322.34	2796.71
CD4	G	305.46	121.09	339.5	341.73	181.96	201.58	197.98	325.06	199.89	239.59
	P	27677.29	11626.81	7724.43	819.04	10296.15	10207.37	7304.25	12608.9	5695.59	11851.99
	C	305.46	121.09	339.5	341.73	181.96	201.58	197.98	325.06	199.89	239.59
CD5	G	105.29	101.78	141.84	420.8	168.73	104.05	162.87	163.82	106.14	123.22
	P	3247.68	1575.05	5175.05	3130.71	2670.49	2084.89	5942.23	4200.14	5018.18	6118.84
	C	12291.6	13875.02	30718.35	52620.65	30018.14	1868.1	25799.91	3678.71	42021.03	8126.81
CD6	G	63.07	1307.26	321.41	212.69	166.69	1558.16	622.71	240.34	899.1	2685.86
	P	63.07	1307.26	321.41	212.69	166.69	1558.16	622.71	240.34	899.1	2685.86
	C	8.6	331.83	296.56	171.27	137.31	536.92	80.79	158.0	231.79	1417.5
CD7	G	352.9	153.04	467.35	500.7	376.49	305.75	194.75	309.35	400.89	807.76
	P	27724.73	2726.6	7852.27	978.01	10490.68	10311.54	7301.01	12593.19	5896.59	12420.17
	C	352.9	153.04	467.35	500.7	376.49	305.75	194.75	309.35	400.89	807.76
LD6	G	106.16	155.49	211.18	496.48	194.54	126.26	182.92	167.89	158.53	206.79
	P	5859.99	3751.54	7847.36	4217.36	4786.78	6821.6	9482.17	6001.28	9239.52	12799.6
	C	106.16	155.49	211.18	496.48	194.54	126.26	182.92	167.89	158.53	206.79
LD7	G	1.74	335.65	138.66	151.37	51.61	718.52	40.1	8.14	104.77	2379.59
	P	1.74	335.65	138.66	151.37	51.61	718.52	40.1	8.14	104.77	2379.59

Continued on next page

Table B.59 – continued from previous page

Query	Alg	Freemium									
		1	2	3	4	5	6	7	8	9	10
	C	1.74	335.65	138.66	151.37	51.61	718.52	40.1	8.14	104.77	2379.59
LD8	G	242.96	309.34	134.88	155.62	241.05	131.52	316.67	284.51	229.79	369.23
	P	242.96	309.34	134.88	155.62	241.05	131.52	316.67	284.51	229.79	369.23
	C	242.96	309.34	134.88	155.62	241.05	131.52	316.67	284.51	229.79	369.23
LD10	G	106.49	47.67	107.24	94.85	236.94	122.11	121.1	122.83	145.62	160.75
	P	106.49	47.67	107.24	205.94	236.94	122.11	121.1	122.83	145.62	160.75
	C	106.49	47.67	107.24	94.85	236.94	122.11	121.1	122.83	145.62	160.75
LD11	G	46.76	90.66	62.68	702.05	389.29	141.93	206.09	118.8	178.97	190.52
	P	46.76	90.66	62.68	702.05	389.29	141.93	206.09	118.8	178.97	190.52
	C	46.76	90.66	62.68	728.87	389.29	141.93	206.09	118.8	178.97	190.52
LS2	G	84.1	109.16	49.14	53.13	106.3	53.3	119.3	102.76	88.53	135.78
	P	84.1	109.16	49.14	53.13	106.3	53.3	119.3	102.76	88.53	135.78
	C	84.1	109.16	49.14	53.13	106.3	53.3	119.3	102.76	88.53	135.78
LS3	G	4101.24	6260.23	1812.23	12803.5	13405.06	23754.44	562.78	20408.65	246.72	4419.28
	P	X	X	X	X	X	X	X	X	X	X
	C	4101.24	6260.23	1812.23	12803.5	13405.06	23754.44	562.78	20408.65	246.72	4419.28
LS4	G	596.39	465.82	2007.44	159.49	230.49	120.31	324.68	242.31	807.08	479.02
	P	596.39	465.82	2007.44	159.49	230.49	120.31	324.68	242.31	807.08	479.02
	C	596.39	465.82	2007.44	159.49	230.49	120.31	324.68	242.31	807.08	479.02
LS5	G	403.17	793.62	296.02	217.69	1753.75	6132.32	372.26	3588.86	229.64	518.28
	P	403.17	793.62	296.02	217.69	1753.75	6132.32	372.26	3588.86	229.64	518.28
	C	403.17	793.62	296.02	217.69	1753.75	6132.32	372.26	3588.86	229.64	518.28
LS6	G	738.09	326.61	5013.79	668.75	1725.21	1143.86	686.01	2677.73	3364.88	334.85
	P	738.09	326.61	5013.79	668.75	1725.21	1143.86	686.01	2677.73	3364.88	334.85
	C	738.09	326.61	5013.79	668.75	1725.21	1143.86	686.01	2677.73	3364.88	334.85
LS7	G	335.57	278.43	147.4	104.43	129.47	1876.21	194.98	187.21	126.0	288.35
	P	335.57	278.43	147.4	104.43	129.47	1876.21	194.98	187.21	126.0	288.35
	C	335.57	278.43	147.4	104.43	129.47	1876.21	194.98	187.21	126.0	288.35

TABLE B.60: Experiment Results of Query Answer Cost with Settings of Per

Query	Alg	Per									
		1	2	3	4	5	6	7	8	9	10
CD1	G	7.65	5.65	9.84	2.72	8.32	5.45	7.55	1.48	2.4	4.09
	P	7.65	5.65	9.84	2.72	8.32	5.45	7.55	1.48	2.4	4.09
	C	7.65	5.65	9.84	2.72	8.32	5.45	7.55	1.48	2.4	4.09
	G	1.22	1.41	1.93	1.27	2.17	1.51	1.99	0.7	1.2	1.43
CD2		Continued on next page									

Continued on next page

Table B.60 – continued from previous page

Query	Alg	Per									
		1	2	3	4	5	6	7	8	9	10
	P	1.22	1.41	1.93	1.27	2.17	1.51	1.99	0.7	1.2	1.43
	C	4060.4	3000.93	5225.77	1447.48	4420.28	2894.21	4007.9	785.82	1274.43	2169.72
CD3	G	1355.76	6183.6	5436.19	9050.22	10146.62	7526.29	9414.87	5010.4	8888.85	8773.51
	P	1355.76	6183.6	5436.19	9043.22	10146.62	7526.29	9414.87	5006.52	8881.97	8773.51
	C	5414.39	9182.73	10659.33	10496.24	14564.13	10418.61	13420.24	5795.42	10161.91	10941.5
CD4	G	12.38	9.74	14.53	6.49	2.27	6.06	11.41	10.59	8.39	7.84
	P	15671.15	11580.77	20166.13	5585.43	17045.65	11165.02	15464.69	3039.31	4920.03	8372.26
	C	12.38	9.74	14.53	6.49	2.27	6.06	11.41	10.59	8.39	7.84
CD5	G	58.81	250.34	222.28	363.47	404.98	302.32	379.77	204.23	357.87	352.92
	P	10550.9	8553.89	12419.34	6039.86	1412.06	5213.75	9989.28	9851.75	7835.68	7099.18
	C	6957.45	31757.43	27914.03	46484.8	52106.04	38652.85	48352.55	25738.03	45657.22	45061.96
CD6	G	4020.26	3377.26	3498.82	465.93	3917.57	3710.72	4860.23	2796.18	1784.7	3609.08
	P	4020.26	122.31	3498.82	465.93	3917.57	3710.72	4860.23	2796.18	1784.7	3609.08
	C	864.53	122.31	1086.29	63.85	458.44	613.71	829.63	619.89	549.68	795.87
CD7	G	259.75	34.59	328.43	19.78	135.32	182.95	249.72	186.93	165.13	240.52
	P	15917.98	3034.12	20479.33	1465.99	17178.1	11341.52	15702.46	3215.55	5076.61	8604.65
	C	259.75	34.59	328.43	19.78	135.32	182.95	249.72	186.93	165.13	240.52
LD6	G	68.7	252.12	234.71	364.57	409.72	309.1	389.24	211.57	364.26	361.89
	P	18739.5	X	20898.37	X	6262.01	11178.18	18726.18	16235.42	12498.85	13764.87
	C	68.7	252.12	234.71	364.57	409.72	309.1	389.24	211.57	364.26	361.89
LD7	G	1699.56	213.2	2148.44	123.35	873.06	1196.87	1633.36	1234.56	1087.47	1582.26
	P	1699.56	213.2	2148.44	123.35	873.06	1196.87	1633.36	1234.56	1087.47	1582.26
	C	1699.56	213.2	2148.44	123.35	873.06	1196.87	1633.36	1234.56	1087.47	1582.26
LD8	G	205.01	163.69	193.66	67.49	135.82	158.47	159.93	170.89	85.55	217.84
	P	205.01	163.69	193.66	67.49	135.82	158.47	159.93	170.89	85.55	217.84
	C	200.95	145.14	177.36	40.34	105.38	135.89	131.68	155.86	58.88	191.52
LD10	G	6.88	25.14	23.14	35.62	41.06	30.3	38.06	19.71	34.92	34.84
	P	6.88	25.14	23.14	35.62	41.06	30.3	38.06	778.95	1266.18	34.84
	C	6.88	25.14	23.14	35.62	41.06	30.3	38.06	19.71	34.92	34.84
LD11	G	156.95	716.8	630.0	1049.29	1176.23	872.5	1091.41	580.91	1030.59	1017.16
	P	156.95	716.8	630.0	1049.29	1176.23	872.5	1091.41	580.91	1030.59	1017.16
	C	160.49	732.96	644.2	1072.94	1202.74	892.16	1116.01	594.0	1053.81	1040.08
LS2	G	10.13	13.75	14.18	12.89	16.98	15.02	17.18	12.75	13.47	18.91
	P	10.13	13.75	14.18	12.89	16.98	15.02	17.18	12.75	13.47	18.91
	C	10.13	13.75	14.18	12.89	16.98	15.02	17.18	12.75	13.47	18.91
LS3	G	26652.34	19507.44	23733.24	5784.84	14443.61	18350.65	17887.41	20868.15	8229.94	25771.65
	P	X	X	X	X	X	X	X	X	X	X

Continued on next page

Table B.60 – continued from previous page

Query	Alg	Per									
		1	2	3	4	5	6	7	8	9	10
	C	26652.34	19507.44	23733.24	5784.84	14443.61	18350.65	17887.41	20868.15	8229.94	25771.65
LS4	G	48.73	591.72	176.95	3728.14	1712.09	273.9	1801.43	1431.55	1445.91	1059.98
	P	48.73	591.72	176.95	3728.14	1712.09	273.9	1801.43	1431.55	1445.91	1059.98
	C	48.73	591.72	176.95	3728.14	1712.09	273.9	1801.43	1431.55	1445.91	1059.98
LS5	G	6883.46	4887.34	5680.23	1523.93	3584.26	4742.78	3940.58	5628.84	1627.68	6513.07
	P	6864.13	5003.62	5693.51	2307.35	3932.34	4766.52	4323.44	5920.82	1929.86	6723.99
	C	6883.46	4887.34	5680.23	1523.93	3584.26	4742.78	3940.58	5628.84	1627.68	6513.07
LS6	G	169.23	1192.16	403.67	7210.38	3337.92	573.02	3521.38	2820.48	2807.16	2111.96
	P	169.23	1192.16	403.67	7210.38	3337.92	573.02	3521.38	2820.48	2807.16	2111.96
	C	169.23	1192.16	403.67	7210.38	3337.92	573.02	3521.38	2820.48	2807.16	2111.96
LS7	G	2342.55	1954.82	2032.83	3079.66	2233.08	1555.04	2634.18	2773.03	1523.35	2776.86
	P	X	1954.82	2032.83	3079.66	2233.08	1555.04	2634.18	2773.03	1523.35	2776.86
	C	2342.55	1954.82	2032.83	3079.66	2233.08	1555.04	2634.18	2773.03	1523.35	2776.86

B.4.2 Time Efficiency

TABLE B.61: Experiment Runtime (msec) of Query Planning Optimization Algorithms with Settings of Flat. C: CostFed. G: Gen-Greedy. S: Sum-Greedy.

Query	Alg	Flat									
		1	2	3	4	5	6	7	8	9	10
Query	Alg	Flat									
		1	2	3	4	5	6	7	8	9	10
CD1	C	1468	1594	1563	1714	1765	1506	1590	1866	1603	1731
	P	1335662	X	1297304	1313197	1310734	1280943	X	1328640	X	1395544
	S	1653	X	2124	1887	1642	1713	X	1373	X	1755
CD2	C	1670	1581	1288	1552	1356	1404	1683	1722	1622	1725
	P	2432	2529	2431	2174	2258	2149	1931	2089	2273	3290
	S	1496	1831	1773	1866	2089	1768	1820	1585	1595	2758
CD3	C	9344	10348	9699	9397	9444	9498	9233	9209	10207	12115
	P	9321	50035	8975	47890	50509	8541	8773	8988	50950	61411
	S	9707	1116468	8498	1133571	1120757	9159	9041	9346	1077383	1153143
CD4	C	3659	3957	3800	3646	3345	3800	3737	3372	3722	3996
	P	60798	55552	53421	3278	53221	3045	2696	3002	3026	7499
	S	1655636	1621809	1689519	3585	1640182	3777	4222	3889	3948	3493
Continued on next page											

Table B.61 – continued from previous page

Query	Alg	Flat									
		1	2	3	4	5	6	7	8	9	10
CD5	C	12044	13102	11942	12749	11380	12730	11429	12033	12431	15179
	P	11566	12273	11711	140527	12610	11643	11417	141265	139140	160398
	S	17995	15501	12334	6774288	14100	12522	13313	6604237	6547982	6569568
CD6	C	313326	307930	310803	310899	311146	308434	309498	309502	295761	308784
	P	226914	4176686	4173192	219991	4144348	231429	4146535	220928	225767	251832
	S	2483026	4051104	4137805	2517058	4186568	2612665	4122793	2662147	2540566	2376840
CD7	C	61521	61388	60922	61448	61225	60822	61743	61275	58172	60535
	P	323652	319661	329981	332500	331351	328167	332281	325790	323795	345935
	S	7658935	7554199	7497625	7713344	7540487	7661156	7503331	7515124	7711905	7511885
LD6	C	55904	55326	56035	55612	55409	55427	55261	52174	55487	56773
	P	190031	190070	X	X	189623	189027	X	139634	X	X
	S	247116	241807	X	X	245070	243000	X	187333	X	X
LD7	C	304235	308282	303890	310497	308593	301765	292055	291010	301918	295071
	P	274244	256591	261614	258333	261547	263150	260405	261731	263682	293162
	S	283168	281217	294422	300389	282805	295005	296394	283955	295082	299731
LD8	C	29319	28947	28855	28920	29354	28918	29192	29616	27990	29429
	P	219655	X	X	420318	X	330613	227851	X	298009	222552
	S	27771	28904	28031	27248	27500	27057	28327	28587	27309	28105
LD10	C	6480	6201	5983	6305	5912	6414	6456	6514	6614	7889
	P	2267	10454	2394	10321	10254	2248	2458	2416	10214	14253
	S	6542	9015	5914	9385	9106	6662	5973	6030	9215	9805
LD11	C	154952	145578	154283	149946	154462	155765	153639	147856	152881	154304
	P	833814	824369	836031	806832	812934	820217	819606	817412	823184	866889
	S	143469	147841	143602	144076	143232	143703	153839	151874	153307	146450
LS2	C	3597	3328	3601	3644	3538	3742	3553	3651	3286	3475
	P	702539	684714	659615	679737	676254	643882	655781	668004	674492	682347
	S	4183	4157	3849	4229	3263	3149	3219	3480	4246	3305
LS3	C	3010720	3032602	3005464	2995381	3006084	3019231	3074190	3100355	3033220	3011684
	P	256948	81730	81763	267909	83738	249166	261053	82983	261359	270325
	S	2842884	3295773	3377157	2814788	3394527	2706882	2725400	3094363	2822232	2869419
LS4	C	4004	3733	3760	4135	4078	4077	4044	3971	4057	4467
	P	3398	2947	79205	76331	77023	3111	2976	76621	3302	87097
	S	4151	4074	1418431	1442369	1450574	3855	3778	1452811	3884	1405229
LS5	C	1029209	986841	1025439	1041208	1036315	1038813	1039223	1040812	1039047	1043625
	P	43299	34772	X	X	X	36970	35764	X	32459	X
	S	1170460	1024872	X	X	X	1238398	1199706	X	1075008	X
	C	24273	24771	24814	24931	25107	25271	24590	24837	24803	24956
LS6		Continued on next page									

Table B.61 – continued from previous page

Query	Alg	Flat									
		1	2	3	4	5	6	7	8	9	10
LS7	P	15820	15407	216207	214423	214218	15532	15779	215918	15250	226271
	S	24071	24278	2479994	2466355	2521679	23395	23926	2466704	24139	2468899
	C	537003	534847	536502	537343	528761	528778	534324	531958	509294	535744
LS7	P	19219	18182	X	X	X	18050	17882	X	17926	X
	S	505597	503761	X	515284	514967	X	517381	503307	515064	X

TABLE B.62: Experiment Runtime (msec) of Query Planning Optimization Algorithms with Settings of Freemium. C: CostFed. G: Gen-Greedy. S: Sum-Greedy.

Query	Alg	Freemium									
		1	2	3	4	5	6	7	8	9	10
Query	Alg	Freemium									
		1	2	3	4	5	6	7	8	9	10
CD1	C	1673	1968	1363	1695	1308	1708	1418	1576	1479	1553
	P	1283970	1300096	1308833	1325197	1330139	1308478	1293982	1274156	1285198	1294649
	S	1495	1768	1675	1576	1680	1767	2019	1632	1932	1674
CD2	C	1632	1334	1657	1747	1792	1721	1766	1671	1773	1739
	P	1628	2117	1686	1870	1699	1581	1606	1714	1605	1873
	S	1287	1461	1100	1534	1409	1546	1096	1373	1396	1453
CD3	C	9273	9403	9552	9638	9444	9535	9549	9327	9649	9264
	P	6986	6442	6404	6645	6878	6693	6815	6350	6619	6562
	S	8585	9347	8806	1164589	8147	9273	9159	9630	9460	8905
CD4	C	3386	3724	3784	3411	3404	3829	3745	3849	3708	3365
	P	41488	41583	41597	41617	41721	41873	41504	42352	41224	41809
	S	3594	3423	3326	3565	3518	3311	3293	3265	4390	3482
CD5	C	12859	11690	11715	12936	12545	11472	11950	12656	12453	13000
	P	108268	110920	111010	107868	107395	107184	110132	110173	108775	107892
	S	54534	50780	56287	59698	49755	54799	53343	49233	54009	50652
CD6	C	305727	295768	310103	312146	307694	310446	296805	298837	309201	311382
	P	163334	164830	162296	160223	162130	162251	162864	164202	163328	171164
	S	1514345	326562	1521033	1524914	324386	330649	337681	320735	319888	336009
CD7	C	59141	61472	58312	61432	60233	61115	60630	60677	61451	58572
	P	365872	365294	361173	361449	362919	365654	365169	372351	363045	366859
	S	59610	59549	56776	60142	465517	416503	56878	414020	56768	56369
LD6	C	55295	55691	58471	55227	52726	52667	55836	55533	52736	55960
	P	2004087	1986250	2010869	2000626	2003976	1781278	2008688	1778440	1999752	1999642
	S	56494	53838	53687	55535	55035	52865	52617	52629	52506	55141

Continued on next page

Table B.63 – continued from previous page

Query	Alg	Per									
		1	2	3	4	5	6	7	8	9	10
		1	2	3	4	5	6	7	8	9	10
CD1	C	1656	1463	2025	1737	1624	1706	1646	1423	1358	1718
	P	1479473	1305625	1278175	1852843	1695230	1308521	1323923	1312700	1759596	1271933
	S	1485	1719	1708	1560	1715	1376	1926	1804	1661	1807
CD2	C	1673	1659	1635	1628	1791	1355	1677	1757	1630	1708
	P	1870	1930	1999	1711	1933	1708	1934	1789	1941	1905
	S	1469	1317	1109	1070	1167	1625	1183	1167	1088	1219
CD3	C	9736	9511	9616	9570	9594	9408	9114	9643	9072	9657
	P	6246	6576	6465	6967	7084	6756	6696	6454	7362	6644
	S	9974	8743	8576	8673	9715	8586	X	X	X	8913
CD4	C	3659	3419	3953	3752	3879	3898	3985	3694	3952	3805
	P	41378	41096	41543	41676	41946	41401	41602	42031	42380	41592
	S	3546	3601	3479	3315	3546	3270	3268	3506	3625	4164
CD5	C	13167	12921	12073	12684	12747	12084	13052	12330	11738	11794
	P	109375	108430	111183	107698	108804	110802	109238	109187	108421	108121
	S	50696	57485	58492	55847	502276	49625	52234	50974	50607	54398
CD6	C	311448	313988	300873	296878	294676	298876	308102	297155	309619	296763
	P	163850	172279	162799	163924	161047	160215	160379	161823	160605	165297
	S	313334	328795	328204	324658	309808	297500	321384	301142	320448	311346
CD7	C	61482	60654	60998	61810	61468	61474	61029	58178	61277	60750
	P	361951	375518	360294	371405	364986	366199	363488	372065	360212	362762
	S	56529	56424	57484	60437	59676	57588	56950	59378	59302	59430
LD6	C	55509	55403	55773	58387	56017	55740	53339	53025	56264	55280
	P	1785687	X	2007376	X	1993479	2005740	2005858	2003679	2009845	1993107
	S	54610	52243	55096	52911	55212	52472	51930	54918	52256	54502
LD7	C	305110	305089	304236	306229	303556	305544	307222	305468	306976	305500
	P	260439	259754	262532	258664	267045	266326	266275	263632	269097	258444
	S	284862	291696	287233	305249	284378	305011	285065	288057	304778	284989
LD8	C	29302	27882	29636	28033	29829	29241	29573	29203	29564	29480
	P	220656	214939	215143	214204	217553	216096	214426	213813	220183	215211
	S	27481	28504	28028	28395	28051	X	X	X	X	27508
LD10	C	6330	5898	6261	5945	6405	6501	6312	5849	6641	5926
	P	2299	2444	2228	2168	2131	2396	2297	10293	9882	2384
	S	5964	6290	6598	5759	6269	6422	5919	6424	5934	6337
LD11	C	147697	147257	154671	155467	155887	147244	157191	154307	153752	153019
	P	824835	815821	818232	815103	834335	803928	803514	796404	811420	828062
	S	147658	153769	151766	153082	162917	153517	154077	153929	155629	153594

Continued on next page

Table B.63 – continued from previous page

Query	Alg	Per									
		1	2	3	4	5	6	7	8	9	10
LS2	C	3636	3283	3567	3253	3665	3622	3552	3519	3634	3457
	P	685993	682323	699014	670520	688418	675098	673632	699827	648709	685975
	S	4079	3781	4167	4167	3436	3451	3634	3901	3751	4090
LS3	C	3007471	3166068	3010483	3035867	3166640	3073983	3068471	3162321	3172088	2987161
	P	X	X	X	X	X	X	X	X	X	X
	S	3234904	3300395	3153132	2720863	2800856	3383711	3356417	3360686	2653045	3228285
LS4	C	4019	3972	3997	4020	4030	3957	4100	4177	3958	3922
	P	3652	3624	3506	3282	3550	3462	3657	3533	3156	3402
	S	4469	3890	4497	3678	4206	3748	3777	3756	3982	4027
LS5	C	1033197	1026460	1039550	1048483	1002712	1032798	1035663	1033483	1050768	992927
	P	35837	35582	35839	35632	35721	35570	35451	35603	36295	35412
	S	5368680	410041	409818	426010	410311	426172	413978	418354	414590	427422
LS6	C	24480	24583	24451	24580	25248	25188	24589	25145	24968	24559
	P	18289	18569	18639	19048	19063	18779	18437	19117	18753	18245
	S	17338	16579	16479	16673	16524	17242	17428	16472	17162	16371
LS7	C	533801	537160	510927	535332	506826	508891	533501	513008	536749	528823
	P	X	18267	18162	18296	17895	18652	17996	18146	18782	18454
	S	500471	499745	499335	516659	514767	501209	497461	X	X	498689

TABLE B.64: Experiment Results of Query Planning Time (msec) with Settings of Flat.
C: CostFed. G: Gen-Greedy. S: Sum-Greedy.

Query	Alg	Flat									
		1	2	3	4	5	6	7	8	9	10
CD1	G	1299118	1285712	1252881	1277072	1266656	1236777	1255081	1255419	1250896	1358671
	P	1335212	X	1296825	1312731	1310308	1280491	X	1328178	X	1395055
	C	88	88	87	89	85	95	98	95	97	90
CD2	G	1613	1675	1127	1594	1597	1421	1461	1443	1613	2157
	P	1506	1656	1542	1332	1397	1244	1113	1179	1426	2217
	C	64	67	61	67	66	74	67	61	64	64
CD3	G	1853	2036	1766	1778	1733	1716	1716	1752	1686	7253
	P	2002	1745	1926	2044	1793	1740	1618	1764	1977	7379
	C	61	62	61	70	62	60	61	60	60	59

Continued on next page

Table B.64 – continued from previous page

[illegible]

Table B.64 – continued from previous page

Query	Alg	Flat									
		1	2	3	4	5	6	7	8	9	10
	P	2241	2099	X	X	X	2346	2104	X	2226	X
	C	156	150	150	147	157	157	165	142	151	189
LS6	G	2121	1749	2075	2013	2062	1798	2147	1907	1971	4507
	P	2099	1995	1759	1908	1740	1767	1948	2015	2065	4732
	C	69	66	65	70	65	67	77	88	66	78
LS7	G	1972	1967	1728	1653	1594	1895	1843	1537	1850	3885
	P	1718	1635	X	X	X	1655	1694	X	1666	X
	C	112	101	92	98	96	100	98	98	97	111

TABLE B.65: Experiment Results of Query Planning Time (msec) with Settings of Freemium. C: CostFed. G: Gen-Greedy. S: Sum-Greedy.

Query	Alg	Freemium									
		1	2	3	4	5	6	7	8	9	10
Query	Alg	Freemium									
		1	2	3	4	5	6	7	8	9	10
CD1	G	1232176	1230170	1243548	1247428	1270280	1271464	1355043	1240393	1257168	1232326
	P	1283501	1299672	1308370	1324721	1329681	1307921	1293530	1273710	1284771	1294184
	C	93	95	87	108	102	102	97	91	87	99
CD2	G	1540	1116	1109	1436	1309	1491	1144	1470	1494	1476
	P	1117	1595	1216	1360	1218	1096	1102	1237	1127	1394
	C	89	66	67	70	63	79	79	64	63	63
CD3	G	1685	1912	2005	2000	1803	1936	1824	1772	1961	1794
	P	1939	1743	1708	1748	1951	1817	1803	1657	1954	1913
	C	63	58	63	63	61	68	60	86	60	64
CD4	G	2101	2239	2284	1850	2218	2372	2029	2294	2363	2601
	P	2190	1925	2248	1995	2220	1966	2017	2253	1961	2218
	C	70	73	71	69	70	81	71	73	70	72
CD5	G	1911	1735	1973	1866	1884	1674	1791	1730	1765	1870
	P	1682	1839	1918	1689	1703	1623	1878	1903	1868	1624
	C	75	74	77	75	73	72	70	76	75	75
CD6	G	158019	152318	155529	154578	152529	156480	155819	154216	155643	159287
	P	160599	162240	159607	157517	159465	159485	160065	161358	160682	168455
	C	61	71	57	67	61	67	61	69	56	55
CD7	G	166023	161647	164376	161356	163978	166147	158892	161993	164560	165134
	P	156630	151806	152431	153062	155200	154687	156201	161978	153105	156672
	C	65	68	76	65	70	67	70	67	71	67

Continued on next page

Table B.65 – continued from previous page

Query	Alg	Freemium									
		1	2	3	4	5	6	7	8	9	10
LD6	G	2069	2410	2403	2479	2375	2372	2419	2381	2401	2400
	P	2294	2143	2273	2041	2339	2152	2170	2042	2179	2075
	C	69	71	72	77	72	67	72	71	75	71
LD7	G	252090	255469	255758	252660	256506	253174	254631	254463	258268	257183
	P	244145	249720	246979	245752	246584	250709	247148	247822	248429	249967
	C	55	55	51	53	53	58	67	55	53	55
LD8	G	216057	216087	218031	216373	217504	222804	218075	215954	216200	218789
	P	219466	220938	222660	218723	218424	218702	213280	213990	216089	224258
	C	90	90	85	85	87	87	85	91	85	91
LD10	G	1608	1667	1641	1639	1598	1352	1564	1598	1592	1474
	P	1589	1265	1603	1604	1413	1395	1397	1359	1649	1669
	C	60	73	59	62	60	61	60	60	62	62
LD11	G	823345	816736	826472	822518	832732	836150	850833	829794	822346	808892
	P	809480	813840	803472	814537	814270	826113	812790	810455	808126	805676
	C	101	103	93	94	105	92	92	93	93	93
LS2	G	669024	665944	669606	698659	684230	671417	692093	698844	705043	661187
	P	675023	686866	683985	691059	661648	661353	681942	655750	667137	661286
	C	167	178	172	161	190	158	174	162	185	180
LS3	G	1793	1841	1882	1799	1556	1471	1567	1421	1781	2038
	P	X	X	X	X	X	X	X	X	X	X
	C	63	62	61	62	65	62	63	71	61	67
LS4	G	2006	2153	2093	2034	2020	2066	2179	2011	1993	1986
	P	1923	2025	1996	2031	2033	2215	2034	2051	2106	2194
	C	73	51	50	58	47	49	53	58	49	52
LS5	G	2085	2084	2286	2126	2361	2527	1973	2257	2099	2019
	P	2042	2281	2151	2054	2327	2703	2166	2075	2061	2286
	C	140	153	153	143	197	151	156	145	157	156
LS6	G	1748	1785	2014	1705	2099	1802	2019	1831	1772	1829
	P	1632	2055	1733	1785	1649	1755	1797	1935	1977	1724
	C	64	65	66	70	66	65	66	65	66	68
LS7	G	1536	1853	1657	1681	1634	1829	1871	1834	1834	1472
	P	1559	1631	1893	1693	1633	1712	1877	1830	1631	1687
	C	96	93	96	95	97	96	95	95	103	102

TABLE B.66: Experiment Results of Query Planning Time (msec) with Settings of Per.
C: CostFed. G: Gen-Greedy. S: Sum-Greedy.

Query	Alg	Per									
		1	2	3	4	5	6	7	8	9	10
Query	Alg	Per									
		1	2	3	4	5	6	7	8	9	10
CD1	G	1276204	1242881	1832894	1852965	1308037	1308920	1306700	1989109	1807333	1224338
	P	1478946	1305152	1277725	1852343	1694724	1308057	1323458	1312232	1759125	1271462
	C	98	86	101	90	89	95	89	90	92	92
CD2	G	1475	1456	1472	1379	1592	1441	1399	1548	1610	1273
	P	1391	1430	1513	1240	1451	1237	1433	1284	1435	1390
	C	68	68	67	66	69	76	64	63	77	75
CD3	G	1777	1768	1961	1987	1990	1615	1782	1668	2006	1782
	P	1745	1924	1932	2098	1965	1896	1904	1688	1748	1761
	C	60	58	63	65	59	58	60	62	63	73
CD4	G	2251	2050	2259	2228	2286	2098	2234	2292	2249	2252
	P	1842	1959	1913	2224	2257	1984	2235	2191	2216	2022
	C	77	74	72	72	73	72	74	72	73	72
CD5	G	1770	1916	1578	1929	1639	1895	1933	1641	2127	1488
	P	1837	1900	1754	1544	1832	1879	1864	1812	1872	1668
	C	73	74	83	71	90	85	82	75	72	74
CD6	G	154981	156914	153317	160102	153295	160609	156862	153471	155356	154370
	P	160951	160717	160053	161207	157613	157530	157560	159145	157797	162595
	C	59	57	71	71	59	58	57	60	58	59
CD7	G	160297	159761	161435	162782	161939	163503	157614	158314	162532	165920
	P	154754	161540	152079	155629	154918	156624	153668	162424	151380	154492
	C	69	69	66	65	67	67	66	71	78	68
LD6	G	2317	2433	2227	2192	2357	2030	2373	2259	2258	2356
	P	2226	X	2339	X	2185	2365	2330	2350	2385	2129
	C	70	70	72	75	79	81	69	68	74	74
LD7	G	253519	251189	259428	255126	252923	253165	257141	255407	258875	255360
	P	248298	245858	250364	244656	254905	256048	254044	250079	257265	244898
	C	53	55	55	56	63	55	62	57	54	54
LD8	G	217439	217097	219053	218102	218818	219220	224886	218999	217986	215884
	P	218694	212991	213256	212004	215649	214150	212263	211597	218299	213324
	C	87	86	97	87	85	91	87	100	87	88
LD10	G	1621	1607	1638	1468	1356	1494	1640	1432	1530	1595
	P	1498	1681	1467	1440	1389	1616	1546	1585	1392	1597
	C	58	60	61	70	60	60	61	58	63	60

Continued on next page

Table B.66 – continued from previous page

Query	Alg	Per									
		1	2	3	4	5	6	7	8	9	10
LD11	G	828273	822609	811481	815890	817229	822432	792313	786904	802246	820288
	P	817386	806685	808573	806066	826120	793868	794000	787585	803167	818535
	C	93	97	96	95	100	93	97	94	92	97
LS2	G	720591	681493	682346	678552	680350	697945	696712	688444	699116	688418
	P	685495	681871	698520	670047	687904	674562	673125	699333	648226	685460
	C	172	176	190	174	184	182	171	175	168	170
LS3	G	1869	1516	1425	1509	1874	1453	1519	1771	1898	1627
	P	X	X	X	X	X	X	X	X	X	X
	C	61	65	69	71	62	84	64	66	64	62
LS4	G	2214	2008	2177	2157	2191	1922	2194	2181	1968	1859
	P	2125	2107	1990	1956	2105	1969	2136	2030	1790	1913
	C	50	49	47	50	47	52	49	50	61	56
LS5	G	2236	2143	2285	2171	2293	2102	2242	2041	2330	2363
	P	2264	2218	2344	2107	2111	2099	2271	2249	2114	2102
	C	163	152	149	145	163	156	151	139	162	153
LS6	G	1821	1753	1639	1979	1643	1739	2127	1897	2201	1744
	P	1801	1772	2008	2014	2022	1733	1775	2124	2019	1794
	C	66	75	72	68	67	74	66	65	66	64
LS7	G	1645	1907	1865	1805	1603	1838	1731	1602	1882	1525
	P	X	1888	1817	1822	1654	1697	1686	1653	1888	1753
	C	94	96	106	101	91	94	110	102	100	103

Appendix C

Illustration of Approximation Algorithm in Chapter 7

C.1 Compare Function of Approximation Algorithm

Algorithm 6: Compare(e, h)

Input: e , an edges of a query graph;

h , an edge of a query graph.

Output: η , the edge with greater weight at the value of z^* .

```

1  $z_{eh} \leftarrow$  where  $w'(e) = w'(h)$ ;
2  $\mathcal{T}_{min} \leftarrow$  MST( $z_{eh}$ , "min");
3  $\mathcal{T}_{max} \leftarrow$  MST( $z_{eh}$ , "max");
4 if  $C(\mathcal{T}_{min}) > B$  then
5    $z^* > z_{eh}$ ;
6   if  $c(e) > c(h)$  then
7      $w^*(e) > w^*(h); \eta \leftarrow e$ ;
8   else
9      $w^*(e) < w^*(h); \eta \leftarrow h$ ;
10  end
11 end
12 if  $C(\mathcal{T}_{max}) < B$  then
13    $z^* < z_{eh}$ ;
14   if  $c(e) > c(h)$  then
15      $w^*(e) < w^*(h); \eta \leftarrow h$ ;
16   else
17      $w^*(e) > w^*(h); \eta \leftarrow e$ ;
18   end
19 end
20 return  $\eta$ ;

```

C.2 Analysis of Query Plan Enumeration

For a query consisting of n triple patterns, a query plan of the query is a full binary tree with n leaves. And the number of different full binary tree with n leaves is $(2n - 2)!/[n!(n - 1)!]$ according to the Catalan number. Furthermore, the different order of triple patterns as leaves also contribute to different query plans even though they have same full binary structure. Hence, the number of different query plans with n triple patterns is

$$P'(n) = \frac{(2n - 2)!}{n!(n - 1)!} \times n! = \frac{(2n - 2)!}{(n - 1)!} \quad (\text{C.1})$$

Besides, due to the number of selected sources for each triple pattern, the variation of choosing sources to collecting triples matching a triple pattern also contribute to the difference of query plans and their costs. Let z denote the number of selected sources of a triple pattern, then the number of query plans that probably end up with different costs are

$$P(n) = P'(n) \times (2^z)^n = \frac{(2n - 2)!}{(n - 1)!} \times 2^{zn}. \quad (\text{C.2})$$

C.3 Query Plan Enumeration Algorithm

Algorithm 7: Query Plan Enumeration, $EQP(tpConList, tpExcList, ssList)$:

Input: $tpConList$, a list of triple patterns; $tpExcList$, a list of triple patterns; $ssList$, a dictionary about selected sources of triple patterns in $tpConList$.

Output: QPs , all query plans of a set of triple patterns.

```

1  $QPs \leftarrow \emptyset$ ;
2 if  $tpExcList.IsEmpty()$  then
3    $aPlan \leftarrow \emptyset$ ;
4   foreach  $tp \in tpConList$  do
5      $aPlan.add(tp, ssList.get(tp))$ ;
6   end
7    $QPs.add(aPlan)$ ;
8   return
9 end
10  $tp \leftarrow tpExcList[0]$ ;
11  $Sours \leftarrow SelectSources(tp)$ ;
12  $Combs \leftarrow PowerSet(Sours)$ ;
13 foreach  $set \in Combs$  do
14    $tpConList' \leftarrow AddTo(tp, tpConList)$ ;
15    $ssList.update(tp, set)$ ;
16    $tpExcList' \leftarrow RemoveFrom(tp, tpExcList)$ ;
17    $EQP(tpConList', tpExcList', ssList)$ ;
18 end

```

References

- Apache jena fuseki. <https://jena.apache.org/documentation/fuseki2/index.html>. Accessed: 2020-11-20.
- Virtuoso sparql query editor. <https://dbpedia.org/sparql>. Accessed: 2020-11-05.
- Towards trusted data sharing: guidance and case studies. Technical report, Royal Academy of Engineering, 2018.
- Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Publishing Company, Inc., USA, 1995.
- Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. Anapsid: an adaptive query processing engine for sparql endpoints. In *International Semantic Web Conference*, pages 18–34. Springer, 2011.
- V. Aggarwal, Y.P. Aneja, and K.P.K. Nair. Minimal spanning tree subject to a side constraint. *Computers & Operations Research*, 9(4):287–296, 1982.
- Rawan Aljamal, Ali El-Mousa, and Fahed Jubair. A comparative review of high-performance computing major cloud service providers. In *2018 9th International Conference on Information and Communication Systems (ICICS)*, pages 181–186, 2018.
- Carlos Buil Aranda and Axel Polleres. Towards equivalences for federated SPARQL queries. In Georg Gottlob and Jorge Pérez, editors, *Proceedings of the 8th Alberto Mendelzon Workshop on Foundations of Data Management, Cartagena de Indias, Colombia, June 4-6, 2014*, volume 1189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014.
- Sarah Berenji Ardestani, Carl Johan Håkansson, Erwin Laure, Ilja Livenson, Pavel Stranák, Emanuel Dima, Dennis Blommesteijn, and Mark van de Sanden. B2share: An Open eScience Data Sharing Platform. In *2015 IEEE 11th International Conference on e-Science*, pages 448–453, August 2015. ISSN: null.
- Aaron A Armstrong and Edmund H Durfee. Mixing and memory: Emergent cooperation in an information marketplace. In *Proceedings International Conference on Multi Agent Systems (Cat. No. 98EX160)*, pages 34–41. IEEE, 1998.
- Douglas B. Taylor and Jennifer Milton. [Towards a new data landscape](#). April 2019.

- Magdalena Balazinska, Bill Howe, Paraschos Koutris, Dan Suciu, and Prasang Upadhyaya. A discussion on pricing relational data. In *In Search of Elegance in the Theory and Practice of Computation*, pages 167–173. Springer, 2013.
- Magdalena Balazinska, Bill Howe, and Dan Suciu. Data Markets in the Cloud: An Opportunity for the Database Community. In *Proceedings of the VLDB Endowment*, volume 4, pages 1482–1485, Seattle, Washington, 2011.
- Stephanie Barrett and Benn Konsynski. Inter-organization information sharing systems. *MIS Q.*, 6(4):93–105, December 1982.
- Richard Bellman and E Stanley Lee. Functional equations in dynamic programming. *Aequationes Mathematicae*, 17(1):1–18, 1978.
- Tim Berners-Lee. [Linked data](#), 2009.
- Otakar Boruvka. O jistém problému minimálním. *Práce Mor. Přírodved. Spol. v Brně (Acta Societ. Scienc. Natur. Moraviae)*, 3(3):37–58, 1926.
- Dan Brickley, Matthew Burgess, and Natasha Noy. Google dataset search: Building a search engine for datasets in an open web ecosystem. In *The World Wide Web Conference, WWW '19*, page 1365–1375, New York, NY, USA, 2019. Association for Computing Machinery.
- Carlos Buil-Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. SPARQL Web-Querying Infrastructure: Ready for Action? In *The Semantic Web – ISWC 2013*, volume 8219, pages 277–293. Berlin, Heidelberg, 2013.
- James P. Callan, Zhihong Lu, and W. Bruce Croft. Searching distributed collections with inference networks. *SIGIR Forum*, 51(2):160–167, August 2017.
- Michael J. Carey, Nicola Onose, and Michalis Petropoulos. Data services. *Communications of the ACM*, 55(6):86, June 2012.
- Angelos Charalambidis, Antonis Troumpoukis, and Stasinos Konstantopoulos. Sema-grow: Optimizing federated sparql queries. In *Proceedings of the 11th International Conference on Semantic Systems*, pages 121–128, Vienna, Austria, 2015. ACM Press.
- Shuchi Chawla, Shaleen Deep, Paraschos Koutris, and Yifeng Teng. Revenue maximization for query pricing. *Proceedings of the VLDB Endowment*, 13(1):1–14, September 2019a.
- Shuchi Chawla, Shaleen Deep, Paraschos Koutrisw, and Yifeng Teng. [Revenue maximization for query pricing](#). *Proceedings of the VLDB Endowment*, 13(1):1–14, September 2019b.
- Boris Chidlovskii and Uwe M. Borghoff. Semantic caching of web queries. *The VLDB Journal*, 9(1):2, 2000.

- Emilia Cioroai, Stanislav Chren, Barbora Buhnova, Thomas Kuhn, and Dimitar Dimitrov. Towards creation of a reference architecture for trust-based digital ecosystems. pages 273–276. ACM Press, 2019.
- Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.
- Richard Cyganiak, Galway David Wood, and Markus Lanthaler. [Rdf 1.1 concepts and abstract syntax](#), 2014.
- Shaul Dar, Michael J Franklin, Bjorn T Jonsson, Divesh Srivastava, Michael Tan, et al. Semantic data caching and replacement. In *VLDB*, volume 96, pages 330–341, 1996.
- Souripriya Das, Seema Sundara, and Richard Cyganiak. [R2rml: Rdb to rdf mapping language](#), 2012.
- Marina De Vos, Sabrina Kirrane, Julian Padget, and Ken Satoh. Odrl policy modelling and compliance checking. In *International Joint Conference on Rules and Reasoning*, pages 36–51. Springer, 2019.
- Shaleen Deep and Paraschos Koutris. [The Design of Arbitrage-Free Data Pricing Schemes](#). In Marc Herbstritt, editor, *20th International Conference on Database Theor.* Leibniz International Proceedings in Informatics, 2017a.
- Shaleen Deep and Paraschos Koutris. Qirana: A framework for scalable query pricing. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 699–713. ACM, 2017b.
- Amol Deshpande and Joseph M. Hellerstein. Decoupled query optimization for federated database systems. In *Proceedings 18th International Conference on Data Engineering*, pages 716–727. IEEE Comput. Soc.
- Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.
- Tao Ding, Liang Liang, Min Yang, and Huaqing Wu. Multiple attribute decision making based on cross-evaluation with uncertain decision parameters. *Mathematical Problems in Engineering*, 2016, 2016.
- Xiaoou Ding, Hongzhi Wang, Dan Zhang, Jianzhong Li, and Hong Gao. A fair data market system with data quality evaluation and repairing recommendation. In *Asia-Pacific web conference*, pages 855–858. Springer, 2015.
- Xin Luna Dong, Barna Saha, and Divesh Srivastava. Less is more: Selecting sources wisely for integration. In *Proceedings of the VLDB Endowment*, volume 6, pages 37–48. VLDB Endowment, 2012.

- Larry A Dunning and Ray Kresman. Privacy preserving data sharing with anonymous id assignment. *IEEE Transactions on Information Forensics and Security*, 8(2):402–413, 2012.
- Schahram Dustdar, Reinhard Pichler, Vadim Savenkov, and Hong-Linh Truong. Quality-aware service-oriented data integration: requirements, state of the art and open challenges. *ACM SIGMOD Record*, 41(1):11–19, 2012.
- Kemele M. Endris, Mikhail Galkin, Ioanna Lytra, Mohamed Nadjib Mami, Maria-Esther Vidal, and Sören Auer. [MULDER: Querying the Linked Data Web by Bridging RDF Molecule Templates](#). In *Database and Expert Systems Applications*, volume 10438, pages 3–18. Cham, 2017.
- Lee Feigenbaum, Gregory Todd Williams, Kendall Grant Clark, and Elias Torres. [Sparql 1.1 protocol](#), 2013.
- Raul Castro Fernandez, Pranav Subramaniam, and Michael J. Franklin. [Data market platforms: trading data assets to solve data problems](#). *Proceedings of the VLDB Endowment*, 13(12):1933–1947, August 2020.
- Stahl Florian, Schomm Fabian, and Vossen Gottfried. Data Marketplaces: An Emerging Species. *Frontiers in Artificial Intelligence and Applications*, pages 145–158, 2014.
- Hector Garcia-Molina. *Database systems: the complete book (2 Edition)*. Pearson Education Inc., New Jersey, 2009.
- Olivier Goldschmidt, David Nehme, and Gang Yu. Note: On the set-union knapsack problem. *Naval Research Logistics (NRL)*, 41(6):833–842, 1994.
- Olaf Görlitz and Steffen Staab. Splendid: Sparql endpoint federation exploiting void descriptions. In *Proceedings of the Second International Conference on Consuming Linked Data-Volume 782*, pages 13–24. CEUR-WS. org, 2011.
- Davide Grande, Jorge Machado, Bryan Petzold, and Marcus Roth. [Reducing data costs without jeopardizing growth](#). June 2020.
- Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 31–40, 2007a.
- Todd J Green, Grigoris Karvounarakis, Nicholas E Taylor, Olivier Biton, Zachary G Ives, and Val Tannen. Orchestra: facilitating collaborative data sharing. 2007b.
- Tobias Grubenmann, Abraham Bernstein, Dmitry Moor, and Sven Seuken. Challenges of source selection in the wod. In *International Semantic Web Conference*, pages 313–328. Springer, 2017a.

- Tobias Grubenmann, Abraham Bernstein, Dmitry Moor, and Sven Seuken. [FedMark: A Marketplace for Federated Data on the Web](#). *arXiv:1808.06298 [cs]*, August 2018a. arXiv: 1808.06298.
- Tobias Grubenmann, Abraham Bernstein, Dmitry Moor, and Sven Seuken. Financing the Web of Data with Delayed-Answer Auctions. In *Proceedings of the 2018 World Wide Web Conference, WWW '18*, pages 1033–1042, Republic and Canton of Geneva, Switzerland, 2018b.
- Tobias Grubenmann, Daniele Dell’Aglia, Abraham Bernstein, Dmitry Moor, and Sven Seuken. Decentralizing the semantic web: Who will pay to realize it? 2017b.
- S.R. Gunn. [Pdf_{late}x instructions](#), 2001.
- S.R. Gunn and C. J. Lovell. Updated templates reference 2, 2011.
- Stefan Hagedorn and Kai-Uwe Sattler. Lodhub—a platform for sharing and integrated processing of linked open data. In *Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on*, pages 260–262. IEEE, 2014.
- Steve Harris, Andy Seaborne, and Eric Prud’hommeaux. [Sparql 1.1 query language](#), 2013.
- Olaf Hartig and M Tamer Ozsu. Linked data query processing. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 1286–1289. IEEE, 2014.
- Ali Hasnain, Qaiser Mehmood, Syeda Sana Zainab, and Aidan Hogan. Sportal: Searching for public sparql endpoints. In *International Semantic Web Conference*, 2016.
- Lars Heling and Maribel Acosta. Cost- and robustness-based query optimization for linked data fragments. In *The Semantic Web – ISWC 2020*, pages 238–257, Cham, 2020. Springer International Publishing.
- Sebastian Henn. Weight-constrained minimum spanning tree problem. 2007.
- Hans Kellerer, Ulrich Pferschy, and David Pisinger. Knapsack problems. 2004, 2003.
- Paraschos Koutris, Prasang Upadhyaya, Magdalena Balazinska, Bill Howe, and Dan Suciu. Toward practical query pricing with querymarket. In *proceedings of the 2013 ACM SIGMOD international conference on management of data*, pages 613–624. ACM, 2013.
- Paraschos Koutris, Prasang Upadhyaya, Magdalena Balazinska, Bill Howe, and Dan Suciu. Query-Based Data Pricing. *Journal of the ACM*, 62(5):1–44, November 2015.
- Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

- Claude Lemaréchal. *Lagrangian Relaxation*, pages 112–156. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. ISBN 978-3-540-45586-8.
- Chao Li, Daniel Yang Li, Gerome Miklau, and Dan Suciu. A theory of pricing private data. *Communications of the ACM*, 60(12):79–86, 2017.
- Chao Li and Gerome Miklau. Pricing aggregate queries in a data marketplace. In *WebDB*, pages 19–24, 2012.
- Yu Li, Eric Lo, Man Lung Yiu, and Wenjian Xu. Query Optimization over Cloud Data Market. In *18th International Conference on Extending Database Technology, EDBT*, pages 229–240, Brussels, 2015.
- Fan Liang, Wei Yu, Dou An, Qingyu Yang, Xinwen Fu, and Wei Zhao. A Survey on Big Data Market: Pricing, Trading and Protection. *IEEE Access*, 6:15132–15154, 2018.
- Ting-Peng Liang and Jin-Shiang Huang. An empirical study on consumer acceptance of products in electronic markets: a transaction cost model. *Decision support systems*, 24(1):29–43, 1998.
- Ee-Peng Lim and Jaideep Srivastava. Query optimization and processing in federated database systems. In *Proceedings of the second international conference on Information and knowledge management*, pages 720–722, 1993.
- Bing-Rong Lin and Daniel Kifer. On arbitrage-free pricing for general data queries. *Proceedings of the VLDB Endowment*, 7(9):757–768, May 2014.
- C. J. Lovell. Updated templates, 2011.
- João Guilherme Martinez, Rosiane de Freitas, and A DA SILVA. On the problem of finding all minimum spanning trees. *Matemática Contemporânea*, 45:97–105, 2017.
- Sameer Mehta, Milind Dawande, Ganesh Janakiraman, and Vijay Mookerjee. How to Sell a Dataset? Pricing Policies for Data Monetization. In *Proceedings of the 2019 ACM Conference on Economics and Computation, EC '19*, New York, NY, USA, June 2019. Association for Computing Machinery.
- Jan Michelfeit, Tomáš Knap, and Martin Nečaský. Linked data integration with conflicts. *arXiv preprint arXiv:1410.7990*, 2014.
- H. Gilbert Miller and Peter Mork. From data to decisions: A value chain for big data. *IT Professional*, 15(1):57–59, 2013.
- Thomas Minier, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. [Intelligent Clients for Replicated Triple Pattern Fragments](#). In *The Semantic Web*, volume 10843, pages 400–414. 2018.

- Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. [Federated SPARQL Queries Processing with Replicated Fragments](#). In *The Semantic Web - ISWC 2015*, volume 9366, pages 36–51. Cham, 2015.
- Dmitry Moor, Sven Seuken, Tobias Grubenmann, and Abraham Bernstein. The design of a combinatorial data market. 2019.
- Alexander Muschalle, Florian Stahl, Alexander Löser, and Gottfried Vossen. Pricing approaches for data markets. In *International workshop on business intelligence for the real-time enterprise.*, volume 154 of *Lecture Notes in Business Information Processing*, pages 129–144. Springer, Berlin, Heidelberg, 2012.
- K Neumann. Dynamic programming basic concepts and applications. In *Optimization in Planning and Operation of Electric Power Systems*, pages 31–56. Springer, 1993.
- Rachana Nget, Yang Cao, and Masatoshi Yoshikawa. How to Balance Privacy and Money through Pricing Mechanism in Personal Data Market. *arXiv:1705.02982 [cs]*, May 2017. arXiv: 1705.02982.
- David Opresnik and Marco Taisch. The value of big data in servitization. *International journal of production economics*, 165:174–184, 2015.
- Ozalp Ozer and Yanchong Zheng. [Trust and trustworthiness](#). *SSRN Electronic Journal*, 2017.
- Georgios Paltoglou, Michail Salampasis, and Maria Satratzemi. Collection-integral source selection for uncooperative distributed information retrieval environments. *Information Sciences*, 180(14):2763–2776.
- Ravali Pochampally, Anish Das Sarma, Xin Luna Dong, Alexandra Meliou, and Divesh Srivastava. Fusing data with correlations. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 433–444. ACM, 2014.
- Allison L. Powell, James C. French, Jamie Callan, Margaret Connell, and Charles L. Viles. The impact of database selection on distributed searching. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '00*, pages 232–239. ACM Press.
- Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.
- Umair Qudus, Muhammad Saleem, Axel-Cyrille Ngonga Ngomo, and Young-koo Lee. An empirical evaluation of cost-based federated sparql query processing engines. *Journal Title*, 1:1–5, 2019.
- Bastian Quilitz and Ulf Leser. Querying Distributed RDF Data Sources with SPARQL. volume 5021, pages 524–538. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

- Nur Aini Rakhmawati, Jürgen Umbrich, Marcel Karnstedt, Ali Hasnain, and Michael Hausenblas. Querying over Federated SPARQL Endpoints —A State of the Art Survey. *arXiv:1306.1723 [cs]*, June 2013. arXiv: 1306.1723.
- Ramamoorthi Ravi and Michel X. Goemans. The constrained minimum spanning tree problem. In *Algorithm Theory — SWAT’96*, volume 1097, pages 66–75. Springer Berlin Heidelberg.
- Mehdi Riahi, Thanasis G Papaioannou, Immanuel Trummer, and Karl Aberer. Utility-driven data acquisition in participatory sensing. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 251–262, 2013.
- Dumitru Roman, Javier Paniagua, Tatiana Tarasova, Georgi Georgiev, Dina Sukhobok, Nikolay Nikolov, and Till Christopher Lech. Prodatamarket: A data marketplace for monetizing linked data. 2017.
- Muhammad Saleem and Axel-Cyrille Ngonga Ngomo. Hibiscus: Hypergraph-based source selection for sparql endpoint federation. In *European semantic web conference*, pages 176–191. Springer, 2014.
- Muhammad Saleem, Axel-Cyrille Ngonga Ngomo, Josiane Xavier Parreira, Helena F. Deus, and Manfred Hauswirth. DAW: Duplicate-AWare federated query processing over the web of data. In Camille Salinesi, Moira C. Norrie, and Óscar Pastor, editors, *Advanced Information Systems Engineering*, volume 7908, pages 574–590. Springer Berlin Heidelberg. Series Title: Lecture Notes in Computer Science.
- Muhammad Saleem, Alexander Potocki, Tommaso Soru, Olaf Hartig, and Axel-Cyrille Ngonga Ngomo. CostFed: Cost-Based Query Optimization for SPARQL Endpoint Federation. *Procedia Computer Science*, 137:163–174, 2018.
- Muhammad Saleem, Yasar Khan, Ali Hasnain, Ivan Ermilov, and Axel-Cyrille Ngonga. [An Evaluation of SPARQL Federation Engines Over Multiple Endpoints](#). 2017.
- Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. Fedbench: A benchmark suite for federated semantic data query processing. In *International Semantic Web Conference*, pages 585–600. Springer, 2011.
- Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. SP²bench: A SPARQL performance benchmark. In *2009 IEEE 25th International Conference on Data Engineering*, pages 222–233. IEEE. ISSN: 1084-4627.
- Fabian Schomm, Florian Stahl, and Gottfried Vossen. Marketplaces for data: an initial survey. *ACM SIGMOD Record*, 42(1):15, May 2013.
- Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *International Semantic Web Conference*, pages 601–616. Springer, 2011.

- Carl Shapiro, Shapiro Carl, Hal R Varian, et al. *Information rules: a strategic guide to the network economy*. Harvard Business Press, 1998.
- Uppala Shivakumar, Vadlamani Ravi, and GR Gangadharan. Ranking cloud services using fuzzy multi-attribute decision making. In *Fuzzy Systems (FUZZ), 2013 IEEE International Conference on*, pages 1–8. IEEE, 2013.
- Florian Stahl, Fabian Schomm, Lara Vomfell, and Gottfried Vossen. Marketplaces for Digital Data: Quo Vadis? *Computer and Information Science*, 10(4):22, October 2017.
- Florian Stahl, Fabian Schomm, and Gottfried Vossen. Data marketplaces: An emerging species. In *DB&IS*, pages 145–158, 2014.
- Florian Stahl, Fabian Schomm, Gottfried Vossen, and Lara Vomfell. A classification framework for data marketplaces. *Vietnam Journal of Computer Science*, 3(3):137–143, 2016.
- Florian Stahl and Gottfried Vossen. Data quality adjustments for pricing on data marketplaces. In *LWA*, pages 419–421, 2015.
- Florian Stahl and Gottfried Vossen. Fair knapsack pricing for data marketplaces. In *East European Conference on Advances in Databases and Information Systems*, pages 46–59. Springer, 2016.
- Ruiming Tang, Huayu Wu, Zhifeng Bao, Stéphane Bressan, and Patrick Valduriez. The price is right - models and algorithms for pricing data. In *Database and Expert Systems Applications*, pages 380–394, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40173-2.
- Harsh Thakkar, Kemele M Endris, Jose M Gimenez-Garcia, Jeremy Debattista, Christoph Lange, and Sören Auer. Are linked datasets fit for open-domain question answering? a quality assessment. In *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics*, pages 1–12, New York, NY, USA, June 2016. ACM.
- Thanadech Thanakornworakij, Raja Nassar, Chokchai Box Leangsuksun, and Mihaela Paun. An economic model for maximizing profit of a cloud service provider. In *2012 Seventh International Conference on Availability, Reliability and Security*, pages 274–279, 2012.
- Hong-Linh Truong and Schahram Dustdar. On analyzing and specifying concerns for data as a service. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pages 87–94. IEEE, 2009.
- Prasang Upadhyaya, Magdalena Balazinska, and Dan Suciu. Price-optimal querying with data apis. *Proceedings of the VLDB Endowment*, 9(14):1695–1706, 2016.

- Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. Triple pattern fragments: A low-cost knowledge graph interface for the web. *Journal of Web Semantics*, 37-38:184 – 206, 2016. ISSN 1570-8268.
- W3C. Sparql query language for rdf. <https://www.w3.org/TR/rdf-sparql-query/>. Accessed: 2020-11-12.
- Shouhong Wang and Norman P Archer. Electronic marketplace definition and classification: literature review and clarifications. *Enterprise Information Systems*, 1(1):89–112, 2007.
- Yue Wang, Alexandra Meliou, and Gerome Miklau. A consumer-centric market for database computation in the cloud. *arXiv preprint arXiv:1609.02104*, 2016.
- Perrin Wright. Counting and constructing minimal spanning trees. *Bulletin of the Institute of Combinatorics and its Applications*, 21:65–76, 1997.
- Ju Wu, Santiago Becerra, Jaime Zuluaga, Stephen Petschulat, Je Kwang Lee, et al. Apparatus and method for providing a data marketplace, November 20 2008. US Patent App. 11/750,993.
- Naïem K. Yeganeh, Shazia Sadiq, and Mohamed A. Sharaf. A framework for data quality aware query systems. *Information Systems*, 46:24–44, 2014.
- Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. Quality assessment for Linked Data: A Survey: A systematic literature review and conceptual framework. *Semantic Web*, 7(1):63–93, March 2015.
- Mengia Zollinger, Cosmin Basca, and Abraham Bernstein. Market-based sparql brokerage with matrix: towards a mechanism for economic welfare growth and incentives for free data provision in the web of data. 2012.
- Anneke Zuiderwijk, Euripides Loukis, Charalampos Alexopoulos, Marijn Janssen, and Keith Jeffery. Elements for the development of an open data marketplace. In *Conference for E-Democracy and Open Government*, pages 309–322, 2014.
- Şebnem Yılmaz Balaman. Chapter 7 - modeling and optimization approaches in design and management of biomass-based production chains. In Şebnem Yılmaz Balaman, editor, *Decision-Making for Biomass-Based Production Chains*, pages 185–236. Academic Press, 2019. ISBN 978-0-12-814278-3.