Workload Matters: A Robust Approach to Physical RDF Database Design

by

Güneş Aluç

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2015

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Recent advances in Information Extraction, Linked Data Management and the Semantic Web have led to a rapid increase in both the volume and the variety of publicly available graph-structured data. As more and more businesses start to capitalize on graph-structured data, data management systems are being exposed to workloads that are far more diverse and dynamic than what they were designed to handle. In particular, most systems rely on a workload-oblivious physical layout with a fixed-schema and are adaptive only if the changes in the schema are minor. Thus, they are unable to perform consistently well across different types of workloads.

This thesis introduces fundamental techniques for supporting diverse and dynamic workloads in RDF data management systems. Instead of assuming anything about the workload upfront, these techniques allow systems to adjust their physical designs as queries are executed. This includes changing the way (i) records are clustered in the storage system, (ii) data are organized and indexed, and (iii) queries are optimized, all at runtime. The thesis proceeds with a discussion of the challenges that have been encountered in implementing these ideas in a proof-of-concept prototype called chameleon-db, and it concludes with a thorough experimental evaluation.

Acknowledgements

This thesis would not have been completed without the support and contributions of the following people:

First and foremost, I would like to thank my supervisor Professor Tamer Özsu for his patience, support and contributions. In particular, I thank him for giving me the freedom to choose my PhD topic and allowing me to develop chameleon-db. I thank him for supporting me, without hesitation, every time I failed—and oh boy, I did fail on many occasions in the past six years. I thank him for the countless brainstorming sessions we had, which ultimately shaped the design of chameleon-db. More importantly, I thank him for being a great supervisor and for teaching me how to become an independent researcher.

I sincerely thank Professor Khuzaima Daudjee and Dr. Olaf Hartig for their feedback on this thesis. Khuzaima has had positive influence on the quality of the experimental evaluations, and Olaf, on the formalisms developed in this thesis.

I would also like to thank Professor Grant Weddell. Grant has been very supportive of my research. Even though we never had a chance to work together, Grant has, in countless ways, encouraged me to succeed.

I would like to thank my committee members, Professor Ihab Ilyas, Professor Wojciech Golab, and Professor Renée J. Miller for their very valuable feedback and helpful questions. The thesis is in a much better shape now thanks to their feedback.

I would also like to thank Gordon Boerke for his patience and technical support every time my PC broke.

I sincerely thank my fellow colleagues at the Database Research Group (now, the Data Systems Group). In particular, I would like to thank Dr. Oğuzhan Özmen, Dr. Jeffrey Pound, Dr. Andrew Kane, Dr. Umar Farooq Minhas, Greg Drzadzewski and Khaled Ammar for the wonderful coffee chats we had. I would also like to thank my friends Professor Murat Cenk, Professor Emre Çelebi, Professor Selçuk Onay and Professor Yelda Türkan, and of course, Uğur, Derya, Olivier, Robert, Ömer and Ümit for their support and encouragement in all these years.

Finally, I would like to thank my family for their support. In particular, I would like to thank my sister Deniz and my brother in-law Magnus for being there for me, and I would like to thank Dragana, Jovanka and Vojislav for supporting me over these years.

My parents Naci and Tülay deserve a special thanks—earning this PhD title has not been easy and it would not have been possible without their love and support, and I dedicate this thesis to them.

The last but not the least, I would like to thank my wife Dr. Bojana Bislimovska. Bojana had to go through the whole process twice, once for her own PhD (and defence), and now, with mine. Her support and love kept me going in all these years—even in those days when I felt like quitting my PhD. Thank you for understanding, supporting and being with me.

Dedication

This thesis is dedicated to my parents Naci and Tülay.

Table of Contents

List of Tables

List of Figures

Nomenclature

$\hat{E}(Q)$	Edges of a basic graph pattern Q
$\hat{V}(Q)$	Vertices of a basic graph pattern Q
$\mathbb{Z}^*$	The set of non-negative integers
$\mathcal{A}$	The set of all possible annotations
$\mathcal{L}$	The set of literals
$\mathcal{M}_G$	The set of all possible edge annotations for RDF graph G
$\mathcal{T}$	The set of triples
$\mathcal{U}$	The set of URIs
$\mathcal{V}$	The set of variables
$E(G)$	Edges of an RDF graph G
$G = (V, E)$	An RDF graph
$P2P$	Peer-to-Peer
$Q = (\hat{V}, \hat{E})$	A basic graph pattern
qid	Query identifier
sid	Matching subgraph identifier
ts	Timestamp
$V(G)$	Vertices of an RDF graph G

CBGP Constrained basic graph pattern

G-by-Q Group-by-Query

BGP Basic Graph Pattern

BSBM Berlin SPARQL Benchmark

CDB chameleon-db

DBSB DBpedia SPARQL Benchmark

GSF Generalized Structural Form

HDFS Hadoop Distributed File System

LDBC Linked Data Benchmark Council

LOD Linked Open Data

LSH Locality-Sensitive Hashing

LUBM Lehigh University Benchmark

OLAP Online Analytical Processing

OLTP Online Transaction Processing

RDF Resource Description Framework

SPARQL SPARQL Protocol and RDF Query Language

URI Uniform Resource Identifier

WatDiv Waterloo SPARQL Diversity Test Suite

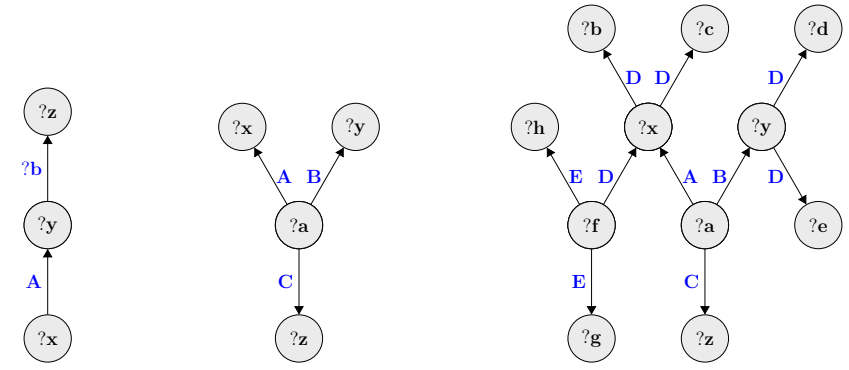
Chapter 1

Introduction

The Resource Description Framework (RDF) is a standard for conceptually describing data on the Web [125, 170], and SPARQL Protocol and RDF Query Language (SPARQL) is the query language for RDF [159].

Advances in the Semantic Web [39, 173] and Linked Data Management [102, 103, 171], together with the proliferation of very large, heterogeneous RDF datasets such as those in the Linked Open Data (LOD) cloud [42, 43] are contributing to the increase in both the quantity and the variety of Web applications that rely on the SPARQL interface to query RDF data [104, 141]. Thus, the demand for high-performance RDF data management systems is increasing. However, despite advances in RDF data management [6, 46, 48, 55, 74, 148, 186, 188, 201], existing systems are still unable to achieve consistently good performance across different types of SPARQL workloads [18].

The problem is that, with increasing diversity in Web applications, workloads that RDF data management systems service are becoming far more *diverse* [30, 43, 71] and far more *dynamic* [124] than what these systems were designed to support [19]. Specifically, (i) web applications that are supported by RDF data management systems are far more varied than conventional relational applications [43], (ii) data that are being handled are far more heterogeneous [71], and (iii) SPARQL is far more flexible in how triple patterns (i.e., the atomic query unit) can be combined [30], which all contribute to *structural* and *data-driven diversity* [18]. For example, different parts of the database can be queried with different query structures all within the same workload, which can be (i) linear (e.g., Figure 1.1a), (ii) star-shaped (e.g., Figure 1.1b), (iii) snowflake-shaped (e.g., Figure 1.1c), or (iv) an even more complex combination of structures, each requiring a different physical



(a) Q_1 (b) Q_2 (c) Q_3

Figure 1.1: Sample SPARQL queries

optimization such as a different type of index.[1] Likewise, some queries in the workload can be very selective while the others involve aggregation and are not selective at all, requiring a completely different physical representation, which is now a well-understood problem for SQL and has led to the introduction of row-stores for Online Transaction Processing (OLTP) and column-stores for Online Analytical Processing (OLAP). Furthermore, hotspots in RDF, which denote the RDF resources that are frequently queried, have fluctuating phases of popularity. For example, an analysis over real SPARQL query logs reveal that during one week intervals before, during and after a conference, the popularity of the RDF resources related to that conference can significantly peak and then drop [124].

Under such diverse and dynamic workloads, queries for which a system performs poorly are inevitable. Moreover, experiments in this thesis demonstrate that these problematic queries may become so frequent in some workloads that the overall performance of the system for that workload is reduced, even causing the system to time-out. To make matters worse, this deficiency has not been thoroughly revealed in performance studies because benchmark workloads do not truly capture this diversity and dynamism [18].

The argument of this thesis is that conventional methods used in physical RDF database design are no longer adequate for the emerging types of SPARQL workloads. Specifically, most of the existing RDF data management systems rely on a *workload-oblivious* design

[1] RDF and SPARQL are described in detail in Chapter 2. For now, it is sufficient to note the following: (i) RDF logically represents data as subject-predicate-object (s, p, o) statements called *triples*, (ii) basic building block of SPARQL queries are *triple patterns* of the form $(\hat{s}, \hat{p}, \hat{o})$, where $\hat{s}$, $\hat{p}$ or $\hat{o}$ can be variables; and (iii) the semantics of SPARQL query evaluation is to find bindings for these variables in the RDF data.

with a *fixed* physical representation, and none of these systems can dynamically update their physical representation or switch to a better one at runtime as the workload changes. Moreover, techniques such as automatic (physical) schema design [10, 11, 37, 62, 92, 135, 198], materialized view selection [56, 86, 94], self-tuning databases [52, 60] and database cracking [95, 113–115] are either offline or scalable at runtime only if the proposed changes in the physical schema of the database are minor. However, the diversity and dynamism in emerging SPARQL workloads may require global changes in the physical schema and the design of the database. For example, it may be necessary to switch from a row-oriented representation to a columnar representation at runtime (i.e., within minutes), but this is not possible with existing solutions [19].

More specifically, existing view materialization techniques [56, 86, 94] are not truly runtime-adaptive. In fact, computing the materialized views alone can take more than half an hour as reported by experiments [86]. Conventional techniques in self-tuning databases [52, 60] are not scalable, either. The problem is that these techniques try to tune the database aggressively. That is, if an index needs to be built, it will be built over the whole table, or if an index needs to be dropped, the whole index will be dropped. In that respect, database cracking [95, 113–115] offers a more scalable solution because tuning is lazy. That is, only very small chunks of the database are tuned at each tuning step. On the other hand, database cracking is applicable only to in-memory arrays in a column-store, and a much wider spectrum of solutions are needed for RDF systems. In contrast, this thesis proposes techniques for *workload-aware* and *runtime-adaptive* RDF data management.

This chapter is organized as follows. Section 1.1 discusses the problems with the physical design of existing RDF data management systems. Section 1.2 presents a vision for developing workload-aware and runtime-adaptive RDF data management systems; and Section 1.3 outlines the scope of this thesis, in particular, the assumptions that are taken, a list of the techniques that have been developed so far to realize this vision, as well as the challenges that are yet to be addressed.

1.1 Problems of Conventional Physical Design

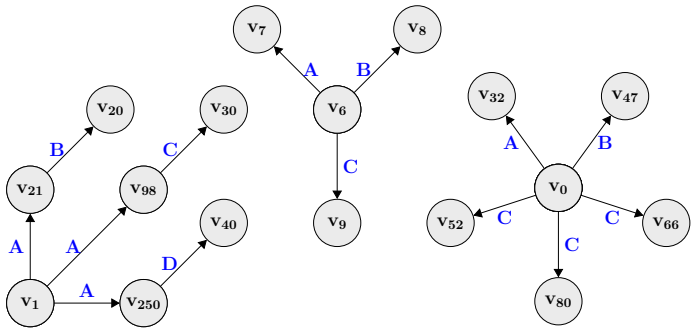
RDF is a schema-free data model in which data are logically represented as subject-predicate-object (s, p, o) statements called *triples* [125] (e.g., Figure 1.2a–1.2b). The RDF data model does not *explicitly* enforce a schema on the data even when the data may be *implicitly* structured. This flexibility makes publishing and linking data across heterogeneous domains much easier, which is an important reason why the LOD cloud has been able to

S	P	O
v_0	A	v_{32}
v_0	B	v_{47}
v_0	C	v_{52}
v_0	C	v_{66}
v_0	C	v_{80}
v_1	A	v_{21}
v_1	A	v_{98}

(a) Dataset

S	P	O
v_1	A	v_{250}
v_6	A	v_7
v_6	B	v_8
v_6	C	v_9
v_{21}	B	v_{20}
v_{98}	C	v_{30}
v_{250}	D	v_{40}

(b) ... continued



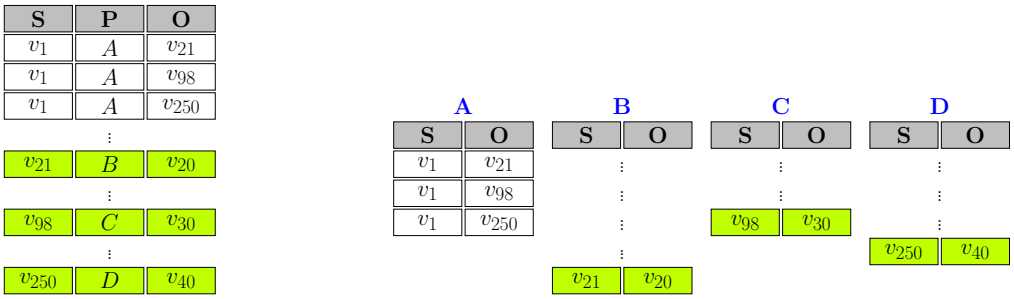
(c) RDF graph

Figure 1.2: Sample RDF Dataset and its Graph Representation

achieve data integration at a very large scale [43]. On the downside, RDF's flexibility comes at a price: without the schema of the data known upfront, physical RDF database design is difficult, which has led to the development of multiple competing physical representations.

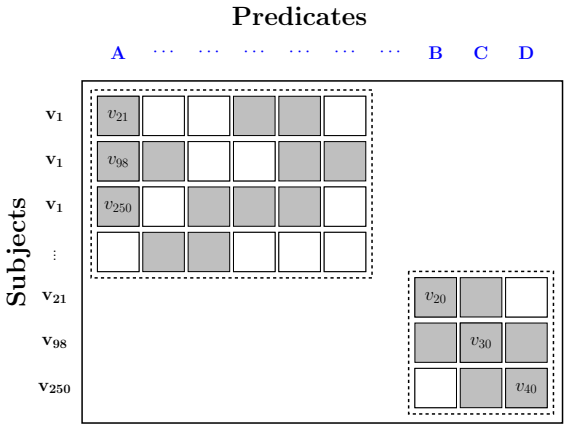
One option of physically representing RDF data is to use a single large table with only three attributes: s, p and o [55] (Figure 1.3a). As a slight variation of this representation, another option is to maintain multiple copies of the table, where each table has a clustered index that implements a different sort-order [34, 100, 148, 186]. It has also been argued that for different workloads, grouping data can provide performance benefits [46, 175, 188]. Therefore, two other representations were developed: (i) **grouping-by-predicates**, where the RDF database is partitioned into two-column tables (one table per predicate) with the tables being stored in a column-store [6, 34] (Figure 1.3b); and (ii) **grouping-by-entities**, where implicit relationships within the data are determined in advance (either as a manual or automated process) to compute a relational schema, and data are mapped to an instantiation of this schema [46, 188] (Figure 1.3c). Another alternative is to rely on the native graph structure of the RDF data [48, 106, 109, 201] (e.g., Figure 1.2c). In this case, **grouping-by-graph vertices**, whereby edges in the RDF graph are grouped based on their incidence on a vertex, is a feasible physical representation (Figure 1.3d).

RDF data management systems, whether single node [6, 34, 46, 48, 55, 74, 100, 148, 186, 188, 201] or distributed [121], rely on one of the above physical representations (or their slight variations); however, all of these representations are *workload-oblivious* and each representation has its shortcomings. Furthermore, none of the existing systems can dynamically switch to a better physical representation at runtime as the workloads change. For systems like *RDF-3x* [148] and *gStore* [200], switching to a new representation is not

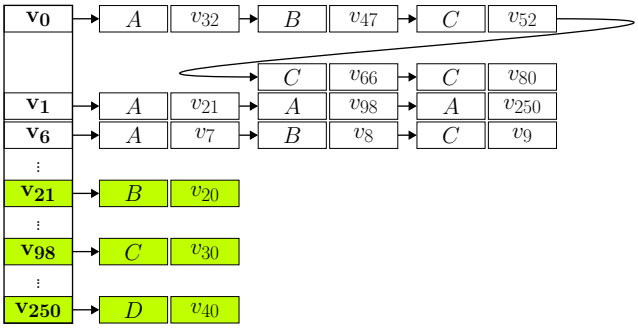


(a) Single table representation

(b) Columnar representation



(c) Relational representation



(d) Adjacency list representation

Figure 1.3: Alternative physical representations of the dataset in Figure 1.2

an option because that would require redesigning and reimplementing the DBMS code from scratch. For systems like *Virtuoso* [74] and *DB2-RDF* [46], which rely on the relational representation, dynamically mapping the data to a new relational schema is not practical due to the overhead of updating the physical design, where conventional self-tuning techniques [52, 60] are not scalable—let alone the fact that these systems cannot switch to a column-oriented physical representation. Consequently, under diverse and dynamic workloads, systems suffer from three major performance barriers: (i) physical data fragmentation, (ii) suboptimal pruning by the indexes, and (iii) processing of unnecessary intermediate result tuples.

1.1.1 Physical Data Fragmentation

In all of the four workload-oblivious representations in Figure 1.3, random I/O due to fragmentation is inevitable for at least some type of queries. Consider Q_1. In the single table representation, the results of the triple pattern $(?y, ?b, ?z)$ in Q_1 are fragmented across the table (i.e., indicated by the colored triples in Figure 1.3a). Since this table is clustered on disk according to attribute S, data are also physically fragmented, which would also be true for any other index that implements a different sort order such as the indexes in RDF-3x. In the columnar representation, these data are fragmented across multiple columns such that they would actually be stored in different files (Figure 1.3b). In the adjacency list representation, vertices v_{21}, v_{98}, v_{250} are fragmented across the hash buckets (Figure 1.3d), hence, similar arguments can be made about physical layout. Whether data relevant to this query are fragmented in the relational representations depends on the physical schema of the database (Figure 1.3c), which is determined, by current techniques [46], based on the entities in the dataset but *not* the workload.

1.1.2 Suboptimal Pruning by the Indexes

How data are grouped together impacts the natural choice of indexing. For example, in the adjacency list representation, triples that are incident on the same vertex are clustered. In other words, if two triples can be joined on their subject-subject (SS) or object-object (OO) attributes, they must also be placed within the same list. SS and OO joins are the only two types of joins necessary for answering star-shaped queries; therefore, an index built across the adjacency list will have a clear advantage for star-shaped queries. As a case study, consider *gStore* [201], which creates a signature entry for each vertex in the adjacency list based on the corresponding values in the list, and stores these signatures in a VS-tree for

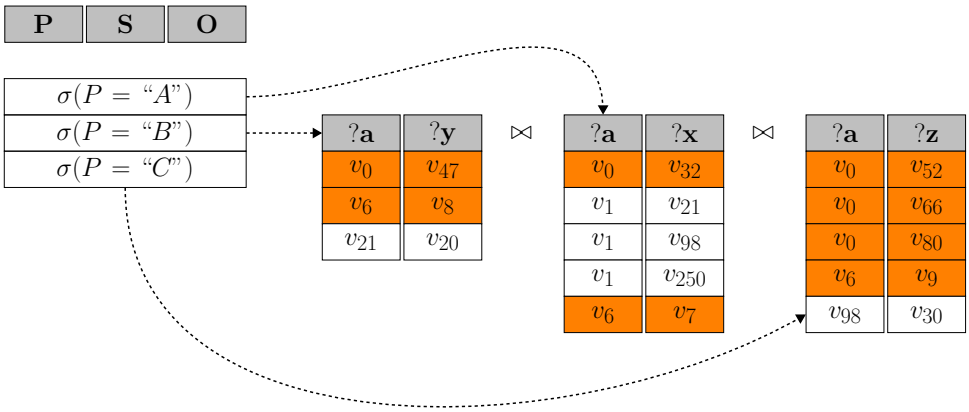
efficient lookup. Then, given a star-shaped query such as Q_2 in Figure 1.1b, which has three bound predicate patterns (i.e., A, B and C), *gStore* can easily locate those vertices whose incident edges satisfy *all* three patterns—in this case, v_0 and v_6 (Figure 1.3d). Contrast this to *RDF-3x* [148] that indexes the large SPO table on six combinations of attributes (i.e., *s-p*, *s-o*, *p-s*, *p-o*, *o-s*, *o-p*), but these indexes can tell only which triples contain A *or* B *or* C. To find out that only the stars centered at v_0 and v_6 satisfy *all* three patterns in the query, the joins need to be computed on the fly (cf., Figure 1.4a).

1.1.3 Processing of Unnecessary Intermediate Result Tuples

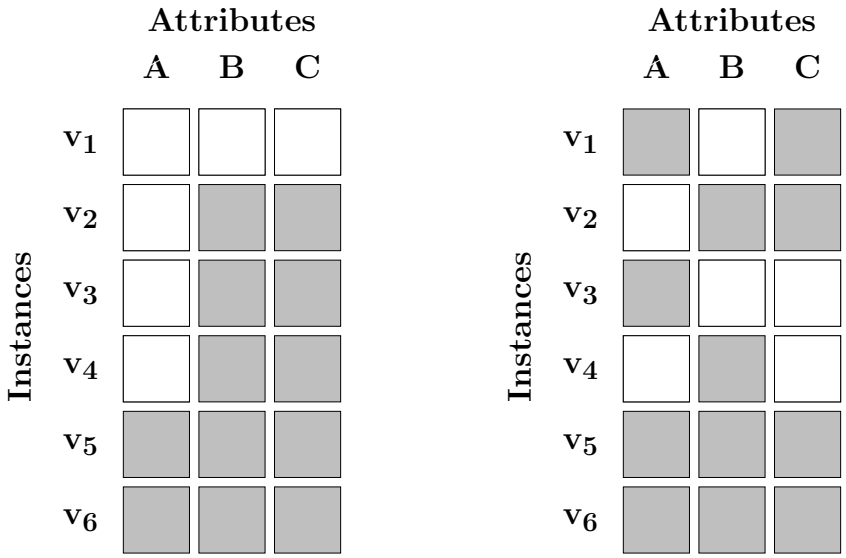
In the relational representations, depending on how data are organized into tables, some queries need to be decomposed, potentially resulting in the processing of unnecessary intermediate tuples. In case of the single table data organization (and its variants used in *RDF-3x*), *every* query needs to be decomposed into its triple patterns, where each triple pattern is evaluated against the large table and the results are self-joined. In this process, some intermediate tuples may not be needed for the computation of the final results. Although runtime optimizations such as sideways information passing [147, 176] exist, they are not always effective. Consider query Q_2 (Figure 1.1b) and the two datasets in Figure 1.4b and Figure 1.4c, where a shaded cell indicates that the value of an attribute (i.e., column) exists for the corresponding instance (i.e., row). For D_1, join operations can be ordered such that the most selective triple pattern (i.e., A) is executed first. This technique, combined with sideways information passing [147], can significantly reduce the sizes of the intermediate result sets. In contrast, for D_2, these optimizations do not work. First, join reordering does not make sense because each triple pattern is equally selective. Second, as demonstrated in Figure 1.4a, sideways information passing will not be effective unless the results from all selection operations are fully processed and the joins are computed.

To quantify the aforementioned problems, an experiment was conducted using the Waterloo SPARQL Diversity Test Suite (WatDiv), which is a benchmark for identifying physical design issues in RDF data management systems [18] (see Chapter 6). Using the WatDiv data generator, 100 million RDF triples, and using the WatDiv stress testing tool, a diverse workload of 12500 SPARQL queries were generated. In the experiment, five popular RDF data management systems are evaluated, namely, RDF-3x [148], MonetDB [112], 4Store [98] and Virtuoso Open Source (VOS) versions 6.1 [76] and 7.1 [74].

RDF-3x follows the single-table approach and creates multiple indexes; MonetDB is a column-store, where RDF data are represented using vertical partitioning [6]; and the last three systems are industrial systems. Both 4Store and VOS group and index data primarily



(a) Self-joins over PSO table: colored intermediate result tuples are necessary and sufficient to compute the final result.



(b) D_1 (c) D_2

Figure 1.4: Self-joins are problematic in the single table representation, potentially generating unnecessary intermediate tuples.

	RDF-3x	VOS [6.1]	VOS [7.1]	MonetDB	4Store
% of queries for which tested system is fastest	20.9%	0.0%	22.6%	56.5%	0.0%
Total workload execution time (hours)	27.1	20.9	20.8	38.6	72.2
Mean (per query) execution time (seconds)	7.8	6.0	6.0	11.1	20.8

Table 1.1: Summary of results over WatDiv 100M RDF triples, 12500 SPARQL queries.

based on RDF predicates, but VOS 6.1 is a row-store whereas VOS 7.1 is a column-store. These systems are configured so that they make as much use of the available main memory as possible.

The results of the experiment can be summarized as follows: (i) no single system performs uniformly well, that is, systems that are fastest are only so for a small percentage of queries in the workload (cf., Table 1.1); and (ii) there can be multiple orders of magnitude difference between the execution times of the fastest and the relatively slower systems (cf., Figure 1.5). Consequently, when the workload is diverse, choosing a suitable system for that workload is a difficult task. One can deploy the system that efficiently executes the most frequent queries in a given workload. However, since the same system can be very inefficient in executing the remaining queries, the overall performance of the system can be far less than optimal. In fact, Table 1.1 illustrates that *none* of the systems that have been evaluated have amortized (i.e., per query) execution times of less than six seconds, which is unacceptable for interactive web applications [145].

1.2 Long-term Vision

It is likely that no single system will ever be the winner for every possible SPARQL query; nevertheless, a much more robust solution can be developed. In particular, a system can be developed that can *automatically* and *continuously* adapt to changing workloads. To achieve this vision, this thesis (i) argues for a **group-by-query** (G-by-Q) representation of RDF data that is purely *workload-driven*, and (ii) proposes ways of **partially tuning**

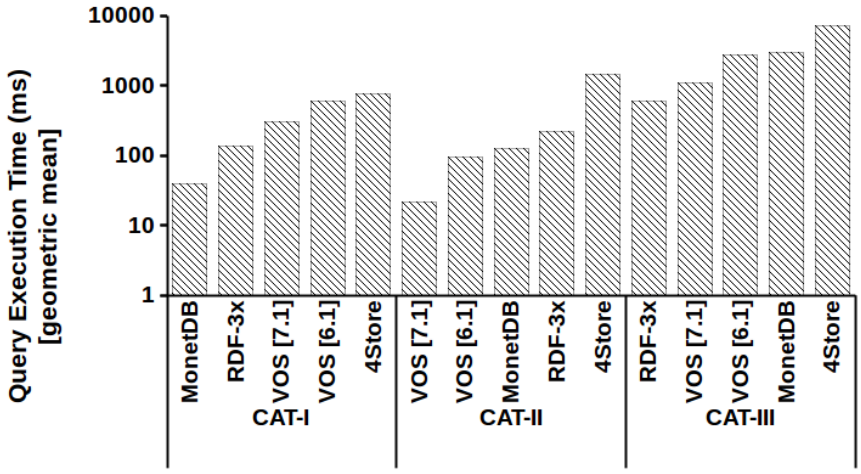


Figure 1.5: Comparison of system performance. CAT-I, CAT-II and CAT-III consist of queries for which, respectively, MonetDB, VOS [7.1] and RDF-3x are the fastest systems (cf., Table 1.1).

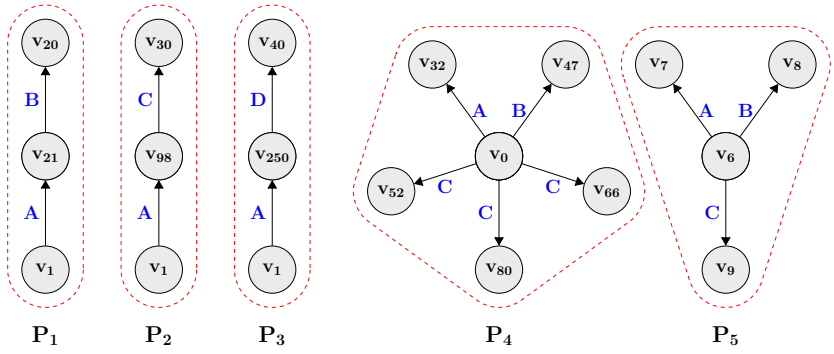


Figure 1.6: Workload-aware group-by-query representation

the database for a lightweight physical design that is easier to update and maintain at runtime.

1.2.1 Group-by-query (G-by-Q) Representation

In the group-by-query (G-by-Q) representation, the content of each database record, as well as the way records are serialized on the storage system are dynamically determined based on the workload. Furthermore, records are *not* necessarily of fixed-length, and grouped into tables. Consider the dataset in Figure 1.2 and queries in Figure 1.1. In the G-by-Q representation in Figure 1.6, different parts of the database are structured based on the different types of queries in the workload (i.e., Q_1, Q_2, Q_3). For example, P_1–P_3 are the three results of the linear query Q_1, and P_4–P_5 are structured based on the results of the star-shaped query Q_2. These records are serialized in the storage system in the order P_1–P_5.

The G-by-Q representation in Figure 1.6 has multiple advantages over the workload-oblivious representations in Figure 1.3. First, queries in the workload can be answered more efficiently, that is, without unnecessary I/O and with better cache utilization because the triples that are needed to compute the results of the queries are already physically clustered (cf., Section 1.1.1). Second, the G-by-Q representation proposes a layout that can be customized for any given workload. Therefore, depending on the workload, not only SS and OO joins but also subject-object (SO) and object-subject (OS) joins will be precomputed and clustered, enabling better indexing opportunities for different query structures (cf., Section 1.1.2). Third, in the proposed representation, data are grouped such that the results of most queries are not segmented, thereby eliminating the need for query decomposition altogether, which was shown to be the culprit for unnecessary intermediate result tuples (cf., Section 1.1.3).

1.2.2 Partial Tuning

For the G-by-Q optimizations to be feasible, the system needs to be able to dynamically update its physical design in a process that does not disrupt normal query execution. In practice, this means that the overhead of each incremental update on the physical design should be very small—given that most queries normally take sub-seconds to be executed, the updates should be much faster, perhaps on the order of microseconds. However, these types of updates are supported by existing self-tuning methods [52, 60] in a scalable way only if the physical schema changes are minor. This may not generally be true in RDF

systems. Consider the single table representation in Figure 1.3a, and assume that data are initially sorted and indexed on attribute O, and that this physical design is suitable for only the first few queries. Hence, at runtime, a tuning advisor decides to switch to the columnar representation in Figure 1.3b; however, to achieve this, the whole physical design needs to be updated from scratch, which is prohibitively expensive. In fact, the table will be partitioned on P and then each partition will be sorted and indexed on S. The former physical design efforts are wasted and this process can take tens of minutes or hours.

This is where *partial tuning* would be beneficial: one can partially cluster and index the database for only the relevant queries, thus preventing a waste of effort when the workload changes (Figure 1.7). For example, one possible design optimization to reduce the I/O overhead in an RDF database can be to co-cluster records on secondary storage based on query access patterns. If this were to be done partially, one could cluster only those records that correspond to the hotspots in the database, leaving the remaining ones unclustered. Then, hot records can be represented in main-memory using adjacency lists or even more sophisticated graph data structures, but for cold records, much less sophisticated data structures would suffice. Indexes can be constructed across the records in such a way that for frequent queries, they return a small number of false-positives (i.e., records that are not relevant to the query) while for the remaining queries they can be less efficient.

The only mature work in self-tuning databases that can be considered as a partial tuning technique is *database cracking* [115]. However, database cracking does not provide a complete solution for this vision because it works only with arrays in the context of in-memory column stores. As discussed earlier, in the context of RDF, column stores are good only for a subset of SPARQL queries. Furthermore, database cracking focuses on indexing; in contrast, data structures and algorithms are needed that support partial tuning for multiple aspects of physical design in RDF.

1.2.3 Proposed System and Challenges

Figure 1.7 is a reference architecture of the proposed system that relies on the G-by-Q clustering and partial tuning techniques. A prototype implementation of this reference architecture within the context of the chameleon-db system is discussed in Section 1.3. There are multiple challenges in implementing this RDF data management system, and these challenges are discussed with respect to each of the three major components of the system, namely, the (i) storage system and cache, (ii) index, and (iii) query engine.

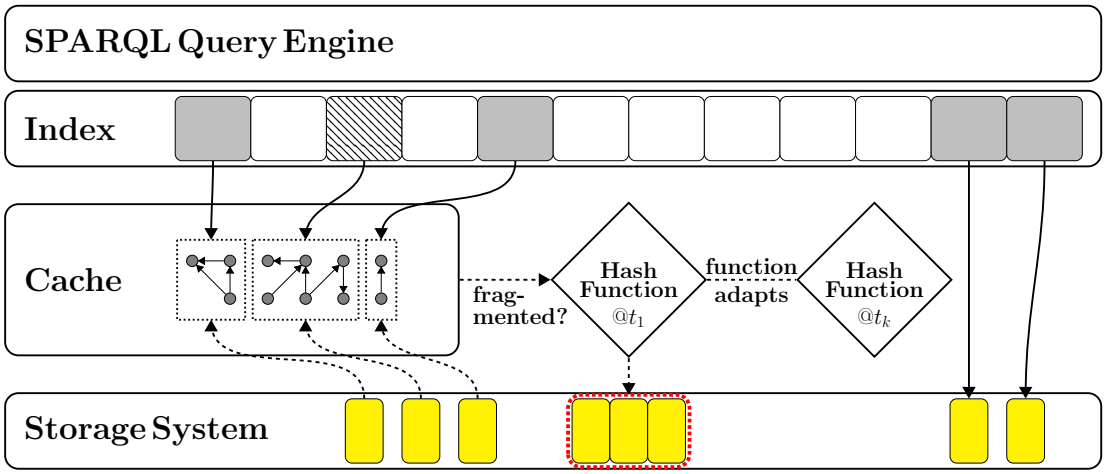


Figure 1.7: Partial indexing and re-structuring of the database

Storage System and Cache

In relation to implementing the G-by-Q approach in the storage system and cache, the answers to two questions are non-trivial:

- Given a SPARQL workload, what is a "good" G-by-Q clustering and how can G-by-Q clusters be materialized in the storage system?
- How can these clusters be efficiently updated when the workload changes?

To answer the first question, one needs to develop measures to quantify the "goodness" of a G-by-Q clustering with respect to a workload. These measures should take into account the performance barriers discussed in Section 1.1, such as physical data fragmentation and processing of unnecessary intermediate results, and the objectives of the G-by-Q clustering should be to minimize these problems. Once a "good" G-by-Q clustering is computed, a one-to-one mapping from the G-by-Q clusters to the physical records in the storage system is possible, where each record can be serialized as a sequence of RDF triples in some predefined order.

As the workload changes, the way data are grouped into records, which represent G-by-Q clusters, may no longer be suitable for the new workload (due to physical data fragmentation, suboptimal pruning by the indexes and/or processing of unnecessary intermediate results), therefore, they need to be updated. Second, access patterns over the storage system will change, indicating that records need to be re-clustered to reduce

random I/O and cache stalls [12]. Clustering algorithms used in conventional database design are not suitable for runtime execution—clustering is NP-hard and approximations have quadratic complexity [118]. Techniques are needed with similar objectives to these clustering algorithms, but with linear running time.

Consider one possible approach that combines hashing with caching (Figure 1.7). Let us assume that initially the database is structured such that each record contains exactly one triple. Likely, this is not a good representation for many workloads; however, as queries are executed, there is an opportunity for partially re-structuring the database by predicting the future relevance of cached records and updating the database as the workload changes. This is an open research problem (i.e., predictive models are discussed below). However, assuming that a good prediction algorithm can be designed, there are various opportunities: (i) multiple records may be packaged/merged into a new record, (ii) a record may be split into multiple records, (iii) records that are co-accessed across multiple queries may be placed contiguously on the storage system, or (iv) records that are no longer co-accessed may be distributed. As shown in Figure 1.7, to quickly determine which records go together, a hash function is used (i.e., records that have the same hash value are assumed to have similar access patterns). The challenge is in designing an adaptive hashing scheme that can auto-tune as the query access patterns change.

Index

Typically, the total number of attributes is much larger in RDF than enterprise relational data. Furthermore, in the G-by-Q representation, records can be (i) variable sized, and (ii) arbitrarily structured. These properties make indexing a non-trivial problem.

Irrespective of the actual implementation (a possible implementation will be discussed in Section 5.5), the proposed index should have the following properties in order to support partial tuning. Consider the abstract index structure in Figure 1.7. Given a query Q, let $\mathbb{I}(Q)$ denote the set of records that are returned by the index (i.e., shaded and striped pointers) and let $\mathbb{R}(Q)$ denote the records that are *truly* relevant to Q (i.e., shaded pointers). As long as, for every possible query Q, the index satisfies $\mathbb{I}(Q) \supseteq \mathbb{R}(Q)$ (i.e., the index may return false-positive records but is guaranteed not to have any false-negatives) correct query results can be produced because false-positives can be eliminated in a validation step during query execution.

In the aforementioned scheme, there is clearly a trade-off between (i) the index construction and maintenance overhead and (ii) the validation overhead. Therefore, the challenge is in designing an index such that for any query that lies within the window of interest

(i.e., recent and frequent queries) it returns as few false-positives as possible whereas for older queries that are no longer frequent, it deliberately returns false-positives. The advantage of this partial indexing scheme is that it is easy to update due to its small size while still being efficient for queries that are currently in use.

Query Engine

The query engine is responsible for parsing SPARQL queries, generating (and optimizing) query plans, and computing the results of queries based on these query plans.

In contrast to conventional (relational) records, clusters in the G-by-Q representation do not have fixed sizes nor contain triples that have the same set of predicates. This schemaless representation enables easy customization of the physical data structures and indexes in the database based on the current workload, resulting in (i) more efficient I/O and cache utilization, (ii) better indexing and data localization, and (iii) fewer intermediate result tuples during query evaluation.

On the other hand, there is a price for this flexibility—generating and executing valid query plans becomes more challenging than it is for fixed, non-adaptable representations. First, one does not have *any* a priori knowledge about how data will be clustered and physically organized in the storage system and within the indexes. All of these decisions depend on the current workload, and as the workload changes, the underlying G-by-Q representation will change as well. This flexibility automatically rules out the possibility of designing and implementing query evaluation code in the DBMS based on a fixed representation. Systems such as RDF-3x [148], MonetDB [112] and gStore [201] are all implemented in this fashion. Second, there is no physical schema to describe the G-by-Q representation (and such a schema is hard to be computed at runtime) that can be used to efficiently generate valid query plans, which relational systems rely on heavily for query plan generation and optimization [94].

1.3 Scope of the Thesis

This thesis address the challenges in Section 1.2.3 by introducing:

1. Measures to quantify the "goodness" of a G-by-Q clustering with respect to a given SPARQL workload;
2. An algorithm to *periodically* compute G-by-Q clusters for a SPARQL workload;

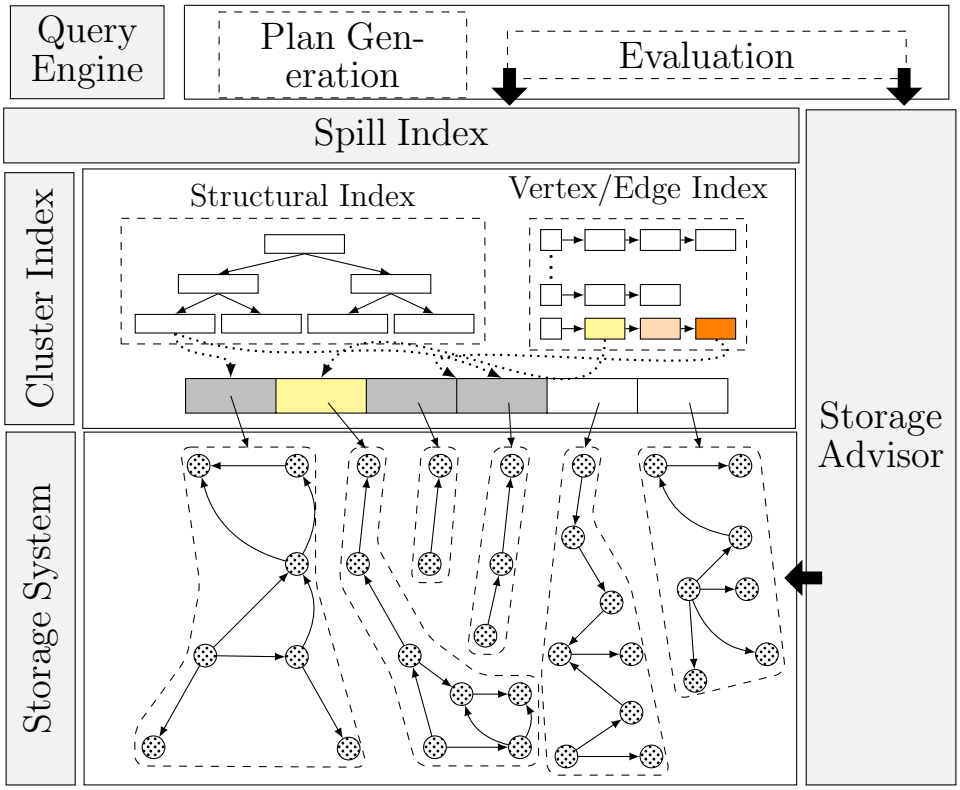


Figure 1.8: Implementation details of chameleon-db

3. An *online* algorithm to compute G-by-Q clusters for dynamically changing SPARQL workloads;

4. A new query evaluation model that can accommodate the dynamic updating of the underlying G-by-Q clusters;

5. Query optimization techniques over the proposed query evaluation model; and

6. A partial, adaptive indexing scheme that is suitable for dynamically updated G-by-Q clusters.

The aforementioned techniques have been implemented in a prototype system called chameleon-db. Figure 1.8 displays the architecture of chameleon-db. The Storage Advisor of chameleon-db is responsible for computing the G-by-Q clusters based on the workload and for periodically updating these clusters, and the Storage System is responsible for

physically storing these G-by-Q clusters on disk. While chameleon-db relies on primarily a disk-based implementation, the system caches frequently accessed G-by-Q clusters in its Buffer Pool. For simplicity, the details of the Buffer Pool have been omitted from Figure 1.8.

The ideas presented in Section 1.2.3 are implemented using two types of indexes, namely, the Spill Index and the Cluster Index. The Spill Index maintains information that is used during query plan generation (and optimization), and the Cluster Index enables the proper G-by-Q clusters to be located during the evaluation of the generated query plan. Various types of indexes have been implemented as part of the Cluster Index, whose details are discussed in Chapter 5.

In chameleon-db, query execution consists of two phases: plan generation (and optimization) and plan evaluation, hence, the corresponding system components in Figure 1.8. Statistics collected during query execution about the underlying G-by-Q clusters are communicated to the Storage Advisor, which uses this information to compute a better G-by-Q clustering next time the physical schema of the database is updated.

Currently, chameleon-db consists of more than $35,000$ lines of C++ code (excluding third-party components such as the SPARQL parser). chameleon-db is used to evaluate the proposed techniques and compare them against techniques or systems that rely on workload-oblivious representations. These evaluations use the Waterloo SPARQL Diversity Test Suite (WatDiv) because, as shown in this thesis, existing SPARQL benchmarks cannot be used for generating diverse and dynamically changing workloads (cf., Chapter 6) [18].

While the techniques presented in this thesis set foundations for achieving the long term vision outlined in Section 1.2, some challenges are beyond the scope of this thesis, and are left as future work.

First, this thesis focuses on the question of "how" to tune but omits the question of "when". The question of "when" to tune the physical design of an RDF data management system is relevant because not all SPARQL workloads may exhibit the same degree of dynamism. Automatically detecting when changes occur in a workload can be an important step to eliminate or reduce redundant tuning steps (hence, the overhead of tuning).

Second, ideally, techniques are needed that can be used for tuning the RDF database after the execution of every query in the workload (i.e., to support extreme dynamism in workloads). While the techniques developed in this thesis are far more scalable than existing solutions (cf., Chapter 6), they support periodic runtime updates such as after the execution of every 10 or 100 queries. Extending these techniques to support more frequent runtime updates is also left as future work.

Third, the techniques presented in this thesis are for scaling-up and for robust query execution across diverse workloads, and it is possible to develop techniques for distribution and scaling-out in the future.

Lastly, the following is an (incomplete) list of other technical problems that remain as future work (which is discussed further in Chapter 7):

1. In the G-by-Q clustering, an RDF triple is the smallest clusterable unit of information; it may be possible to break down RDF triples further.

2. When serializing a G-by-Q cluster on the storage system, each cluster is treated as a sequence of RDF triples sorted on the subjects of the triples, without concern for compression and/or different orderings of the triples.

3. When a record that corresponds to a G-by-Q cluster is brought into the cache, it is represented as an adjacency list—more sophisticated representations are possible, but beyond the scope of this thesis.

4. The techniques proposed in this thesis assume at least some predictability in workloads; more sophisticated predictive models (e.g., those that incorporate oscillations) can be developed in the future.

5. Potentially multiple types of indexes can be developed that implement the abstract partial indexing scheme discussed in Section 1.2.3; however, in this thesis, only a handful are considered.

6. The query engine handles a subset of SPARQL queries, known as basic graph patterns (BGPs); techniques and optimizations beyond BGPs can be developed in the future.

This thesis is organized as follows. Chapter 2 discusses background and preliminaries. Chapter 3 discusses related work. In Chapter 4, the *periodic* and *online* clustering algorithms are introduced. In Chapter 5, query evaluation and indexing techniques are discussed. Chapter 6 reports experimental evaluation of the techniques presented in the thesis, and Chapter 7 concludes with a more detailed outline of future work.

Chapter 2

Background and Preliminaries

In this thesis, both RDF data and the conjunctive fragment of SPARQL queries [159], which is called *basic graph patterns* (BGPs), are represented as graphs; and query evaluation is modeled as a subgraph homomorphism problem [123, 183]. Therefore, for the most part, the standard formalization of SPARQL [157] is relied upon, and only the concepts that are necessary to capture subgraph homomorphism, as it is used in evaluating BGPs over RDF graphs, are introduced. The equivalence between the standard formalization of SPARQL [157, 168] and the one introduced herein is proven in [20].

Assume two disjoint, countably infinite sets of URIs ($\mathcal{U}$) and literals ($\mathcal{L}$), blank nodes are not supported in this formalization. URIs uniquely denote Web resources or features of Web resources. Literals denote values such as strings, natural numbers and booleans. Then, an *RDF triple* is a 3-tuple from the set $\mathcal{T} = \mathcal{U} \times \mathcal{U} \times \big(\mathcal{U} \cup \mathcal{L}\big)$, where for each triple $(s, p, o) \in \mathcal{T}$,

- s is the subject of the triple,
- p is the predicate, and
- o is the object.

Definition 1. *An* RDF graph *is a directed, labeled multi-graph $G = (V, E)$ where:*

(a) the vertices (V) are finite set of URIs or literals such that $V \subset (\mathcal{U} \cup \mathcal{L})$;

(b) the directed, labeled edges (E) are finite set of RDF triples such that $E \subset \big(V \times \mathcal{U} \times V\big) \cap \mathcal{T}$; and

(a) RDF dataset (b) Basic graph pattern

Figure 2.1: Sample dataset and query

(c) each vertex $v \in V$ appears in at least one edge, where for each edge $(s, p, o) \in E$,

- *s is the source of the edge,*
- *p is the label, and*
- *o is the target of the edge.*

Hereafter, $V(G)$ and $E(G)$ are utilized to denote the set of vertices and the set of edges of an RDF graph, respectively. □

Example 1. *The RDF dataset in Figure 2.1a can be formally represented as $G = (V, E)$ where:*

$$V = \{v_1, v_{20}, v_{21}, v_{30}, v_{40}, v_{98}, v_{250}\} \text{ and}$$
$$E = \{(v_1, A, v_{21}), (v_1, A, v_{98}), (v_1, A, v_{250}),$$
$$(v_{21}, B, v_{20}), (v_{98}, C, v_{30}), (v_{250}, D, v_{40})\}$$ □

To define basic graph patterns, a countably infinite set of variables $\mathcal{V}$ that is disjoint from both $\mathcal{U}$ and $\mathcal{L}$ is introduced. As a convention, variables are always preceded by the question mark symbol (?). Similar to RDF graphs, BGPs have a graph-based representation.

Definition 2. *A* basic graph pattern (BGP) *is a directed, labeled multi-graph $Q = (\hat{V}, \hat{E})$ where:*

(a) the vertices $(\hat{V})$ are variables, URIs, or literals such that $\hat{V} \subset \mathcal{V} \cup \mathcal{U} \cup \mathcal{L}$;

(b) the directed, labeled edges ($\hat{E}$) *are 3-tuples such that* $\hat{E} \subset \hat{V} \times (\mathcal{V} \cup \mathcal{U}) \times \hat{V}$*, where for each edge* $(\hat{s}, \hat{p}, \hat{o}) \in \hat{E}$*,*

- $\hat{s}$ *is the source of the edge,*
- $\hat{p}$ *is the label, and*
- $\hat{o}$ *is the target of the edge; and*

(c) each vertex $\hat{v} \in \hat{V}$ *appears in at least one edge.*

Hereafter, $\hat{V}(G)$ *and* $\hat{E}(G)$ *are utilized to denote the set of vertices and the set of edges of a BGP, respectively.* □

Example 2. *The basic graph pattern in Figure 2.1b can be formally represented as* $Q = (\hat{V}, \hat{E})$ *where:*

$$\hat{V} = \{?x, ?y, ?z\}$$
$$\hat{E} = \{(?x, A, ?y), (?y, ?b, ?z)\}$$ □

Note that each edge in a BGP represents a triple pattern (cf., standard formalism in [157]). Therefore, depending on the context, these two terms can be used interchangeably. Each triple pattern consists of three parts, namely, the subject, predicate and object of the triple pattern [157–159]. If the subject, predicate or object of a triple pattern is a variable, it is called an *unbound* pattern, otherwise, the pattern is *bound* [158, 159].

The only deviation from the standard formalism [157] is in the way solution mappings are defined for BGPs. That is, for BGPs, solution mappings are computed from subgraphs of a queried RDF graph that match the BGP. To accommodate this difference, first, the notion of *compatibility* is introduced, which is defined between an edge in an RDF graph and an edge in a BGP (Definition 3). Informally, two edges are compatible if they have the potential to match. Formally:

Definition 3. *Let* $e = (s, p, o) \in E$ *be an edge in an RDF graph* $G = (V, E)$*, and let* $\hat{e} = (\hat{s}, \hat{p}, \hat{o}) \in \hat{E}$ *be an edge in a BGP* $Q = (\hat{V}, \hat{E})$*. Edges* e *and* $\hat{e}$ *are* compatible *if either*

- $p = \hat{p}$*, or*
- $\hat{p} \in \mathcal{V}$*.* □

Example 3. *Let $G = (V, E)$ denote the RDF graph in Figure 2.1a, and let $Q = (\hat{V}, \hat{E})$ denote the BGP in Figure 2.1b. The edge $(?x, A, ?y) \in \hat{E}$ is compatible with only three edges in E, namely, $(v_1, A, v_{21}), (v_1, A, v_{98}), (v_1, A, v_{250})$, because only these three have the same label as $(?x, A, ?y)$. In contrast, the edge $(?y, ?b, ?z) \in \hat{E}$ is compatible with all of the edges in E because the label $?b$ is a variable.* □

Using the notion of edge compatibility, a match between a BGP and an RDF graph can be defined as surjection from the edges (and vertices) of a BGP onto the edges (and vertices) of an RDF graph (possibly a subgraph of the queried RDF graph) such that corresponding edges are compatible and the source (and the target) vertices of a pair of corresponding edges are also mapped onto.

Definition 4. *Let $G = (V, E)$ be an RDF graph, and let $Q = (\hat{V}, \hat{E})$ be a BGP. Given a solution mapping μ, G μ-*matches *Q if*

(a) $\mathrm{dom}(\mu)$ *is the set of variables mentioned in Q, and*

(b) there exist two surjective functions $M_V : \hat{V} \to V$ and $M_E : \hat{E} \to E$ such that:

- *for each $(\hat{v}_1, v_2) \in \hat{V} \times V$ with $M_V(\hat{v}_1) = v_2$:*
  - *if $\hat{v}_1 \in \mathcal{V}$, then $\mu(\hat{v}_1) = v_2$,*
  - *else $\hat{v}_1 = v_2$;*
- *for each $(\hat{e}_1, e_2) \in \hat{E} \times E$ with $M_E(\hat{e}_1) = e_2$, where $\hat{e}_1 = (\hat{s}_1, \hat{p}_1, \hat{o}_1)$ and $e_2 = (s_2, p_2, o_2)$:*
  - *$\hat{e}_1$ and e_2 are compatible and if $\hat{p}_1 \in \mathcal{V}$, then $p_2 = \mu(\hat{p}_1)$,*
  - *$M_V(\hat{s}_1) = s_2$, and*
  - *$M_V(\hat{o}_1) = o_2$.*

G matches Q *if there exists a solution mapping μ such that G μ-matches Q.* □

Example 4. *As shown in Figure 2.2, the colored RDF graph μ-matches the BGP $?x \xrightarrow{A} ?y \xrightarrow{?b} ?z$ with $\mu(?b) = B$.* □

Putting it all together, the expected result of evaluating a BGP over an RDF graph can be defined as follows.

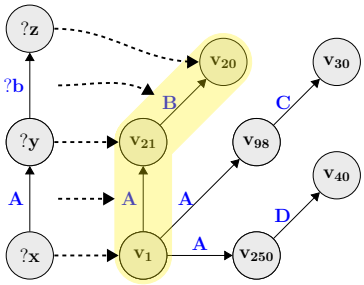


Figure 2.2: An example of a matching subgraph

Definition 5. *The* result *of a BGP Q over an RDF graph $G = (V, E)$, denoted by $[\![Q]\!]_G$, is defined as $[\![Q]\!]_G = \{\mu \mid G' \text{ is a subgraph of } G \text{ and } G' \ \mu\text{-matches } Q\}$. Hereafter, each aforementioned G' such that G' μ-matches Q, is called a* matching subgraph *of G with respect to Q.* □

BGPs can be combined using operators AND, UNION, and OPT [157]. Thus, any BGP is a *SPARQL query*, and if S_1 and S_2 are SPARQL queries, and F is a filter expression, then expressions $(S_1 \text{ AND } S_2)$, $(S_1 \text{ UNION } S_2)$, $(S_1 \text{ OPT } S_2)$, and $(S_1 \text{ FILTER } F)$ are also SPARQL queries. The semantics of these queries can be defined using the standard formalism [157], where solution mappings are combined or manipulated using union ($\cup$), join ($\bowtie$), difference ($\setminus$) and selection (Θ).

The algorithms and data structures introduced in this thesis are optimized for BGPs, and more complex fragments of SPARQL are beyond scope. Therefore, in the remainder of this thesis, the focus will be on Definition 5, which defines the query result over all subgraphs of an RDF graph that match a BGP. Nevertheless, the prototype implementation that was developed (i.e., chameleon-db) can handle a large subset of the SPARQL 1.0 specification (except for complex filter expressions involving built-in functions), and additionally push filter expressions down into BGPs [168]. For the implementation of joins ($\bowtie$), unions ($\cup$) and set difference ($\setminus$), existing techniques were slightly adapted and utilized [148].

Chapter 3

Related Work

This chapter studies existing RDF data management systems across four major dimensions: (i) each system's choice of physical layout (Section 3.1), (ii) the indexing techniques used in the system (Section 3.2), (iii) query evaluation and optimization strategies used in the system (Section 3.3), and (iv) techniques for scaling-out (Section 3.4). The chapter also includes a discussion on existing techniques for evaluating RDF data management systems (Section 3.5) as well as on locality-sensitive hashing (Section 3.6).

3.1 Choice of Physical Layout

Physical design has been the topic of ongoing discussion in the world of RDF and SPARQL [6, 19, 175, 188]. In this thesis, based on their choice of physical design, existing systems are classified into two: (i) systems with a *workload-oblivious* physical layout, and (ii) systems with a *workload-aware* physical layout.

For workload-oblivious physical layouts, one option is to represent RDF data in a single large table with only three attributes: s, p and o [55]. As a slight variation of this representation, another option is to maintain multiple copies of the table, where each table has a clustered index that implements a different sort-order [100, 148, 186]. It has also been argued that for different workloads, grouping data can provide performance benefits [46, 175, 188]. Therefore, two other representations were developed: (i) in the **group-by-predicate** representation (commonly known as vertical partitioning), the RDF database is partitioned into 2-column tables (one table per predicate) and the tables are stored in a column-store [6]; and (ii) in the **group-by-entity** representation (commonly known as

the property table approach), implicit relationships within the data are determined (either as a manual or automated process) to compute a relational schema, and data are mapped to an instantiation of this schema [46, 188]. Another alternative is to rely on the native graph structure of RDF data [48, 106, 109, 201]. In this case, edges in the RDF graph can be grouped based on their incidence on a vertex (i.e., **group-by-vertex**).

The aforementioned physical representations are *workload-oblivious* and can lead to suboptimal execution times for different types of queries, due to reasons such as fragmented data, unnecessarily large intermediate result tuples generated during query evaluation and/or suboptimal pruning by the indexes [19]. To address some of these issues, *workload-aware* techniques have been proposed [19, 20, 56, 86, 107]. Workload-aware techniques can be further classified into three: (i) systems in which the base layout (i.e., base tables) is workload-oblivious, but materialized views are generated based on the workload [56, 86], (ii) distributed systems in which data are distributed across compute-nodes based on a workload, but the physical layout within each node is fixed in a workload-agnostic way [13, 82, 97, 106, 154, 155], and (iii) systems in which the base layout is workload-aware (i.e., the topic of this thesis). The last option is the one advocated in this thesis, where, the **group-by-query** representation is employed.

For example, view materialization techniques have been implemented for RDF over relational engines [56, 86]. However, these materialized views are difficult to adapt to changing workloads. Workload-aware distribution techniques have also been developed for RDF [107] and implemented in systems such as WARP [107] and Partout [82].

3.1.1 Space of Solutions

- **Workload-oblivious representations**:
  - **Single-table** [51, 55, 75, 98, 100, 101, 146, 148, 149, 186, 189]:
    The earliest RDF data management systems [51, 55, 100, 101, 189] rely on the single-table representation, where triples are stored in a single large table with three attributes, corresponding to subject, predicate and object, hence, the name, *triplestore*. Later on, this concept has been extended such that the table includes a fourth attribute [98] to represent the subgraph URI that a set of triples belongs to, to accommodate extensions to RDF [54]. These early systems may have indexes over a few combination of attributes [75], but, the choice of indexing is primarily based on *a priori* assumptions about the workload, which may be speculative.

The first breakthrough came with the development of Hexastore [186] and RDF-3x [146, 148, 149]. These systems argue that ad-hoc choice of indexing does not guarantee robust execution of queries [146, 148, 149, 186], and they proposed indexes to be built across *all* binary permutations of attributes. That is, six different indexes are created for copies of the triples, where each index is clustered (i.e., sorted) on SP, SO, OS, OP, PS or PO.

While experiments demonstrate that these systems are far more robust than the earlier triplestores [146, 148, 149, 186], their major problem is that queries still need to get decomposed into triple patterns, which may result in the generation of intermediate results that are computed from fragmented portions of the clustered indexes [19]. As discussed in Chapter 1, query decomposition might introduce additional problems such as processing of irrelevant intermediate results and suboptimal utilization of the indexes. In contrast, the techniques proposed in this thesis group data based on the queries in the workload, thus, eliminating the need for query decomposition altogether, and/or minimizing the cardinality of intermediate results.

– **Group-by-predicates** [4, 6, 34, 74, 175, 192]: It has been argued that grouping RDF data can have advantages for various types of workloads [4, 6, 34, 74, 175, 188, 192]. One way of grouping data is based on the predicates in the dataset. Abadi et al. [4, 6] proposed a layout for RDF, where a two-attribute table is created for each predicate in the dataset, and the values of subjects and objects associated with that predicate are stored in that table (cf., Figure 1.3b). This representation can be mapped to a column-store producing further advantages (the research prototype C-Store [178] was used in the experiments of the original paper [4], but, in the experiments of this thesis, a much more robust column-store, namely, MonetDB [112] is used).

  For queries in which the predicates to be matched are known in advance (i.e., bound), the aforementioned approach has advantages. On the other hand, in the worst case when there is an unbound predicate pattern in the query, the whole database needs to be searched, which is problematic. Furthermore, even when the predicate patterns in the query are known (i.e., bound), some tables can be very large (due to skewed data distributions), in which case, queries can touch fragmented portions of the tables. The group-by-query representation described in this thesis aims to overcome these limitations.

– **Group-by-entities** [46, 179, 188]: Another way of grouping data is based on the entities in the dataset [46, 179, 188]. In this approach, first, the entity sets are discovered through either a manual [188] or an automated [46, 179]

process, and the data are mapped to a relational database, thus exploiting the performance and optimization power of relational engines. This layout was first used in Jena [188], where a database administrator maps from RDF to a relational schema. The experiments by Sidirourgos et al. [175] demonstrate that there is no sole winner between the group-by-entities representation and the group-by-predicates representation—that, it all depends on the use case (i.e., workload) and the underlying engine used in the experiments. The group-by-query representation that is exploited in this thesis allows data in different parts of the database to be grouped/clustered based on different queries in the workload, thus, better balancing the trade-offs evaluated by Sidirourgos et al. [175].

A second approach in discovering/computing what constitutes an "entity set" is through an automated process. Given a sample dataset, Bornea et al. [46] use a graph coloring scheme to determine the set of entity sets in the dataset, and then use this information to map the data to a relational database. To allow for variability in the structure of the dataset, (i.e., instances of an entity set do not need to share exactly the same set of predicates [71]) the predicates associated with a given instance of an entity are stored as values in the tables, as opposed to column names. Then, given a predicate, a hashing scheme is used to compute the index of the column that stores that predicate (and the associated subject value) [46]. For entities that have multiple values for a given predicate, extra steps are taken. While this representation has been demonstrated to be more efficient with respect to handling NULL values compared to the property table approach, and more efficient in handling star-shaped queries [46], the process of discovering the entity sets is based on the structure of the data but not the queries in the workload. In contrast, in the group-by-query representation, the physical layout is determined purely based on the workload.

– **Group-by-vertices** [76, 109, 156, 182, 190, 200, 201]: The RDF data model is used for representing graph-structured data, and a natural way to (physically) represent graph-structured data is using adjacency lists [200, 201] (matrix representation is also possible [34, 192] but is not as common). In case of the adjacency-list representation, data are naturally grouped based on the vertices in the graph (hence, the name group-by-vertices). Multiple systems [76, 109, 156, 182, 190, 200, 201] exploit this representation where triples that are incident on the same vertex are clustered. In other words, if two triples can be joined on their subject-subject (SS) or object-object (OO) attributes, they must also be placed within the same list. Since SS and OO joins are the only two building

blocks of star-shaped queries, this representation has advantages for star-shaped queries.

However, it has been demonstrated that for other types of queries, such as linear or snowflake-shaped, this representation can still run into performance problems due to fragmentation, irrelevant intermediate result tuples and/or suboptimal pruning by the indexes [18, 19]. The group-by-query representation aims to address these issues.

- **Workload-aware representations**:

  As noted earlier, workload-oblivious physical representations can have problems due to fragmentation, unnecessary intermediate result tuples and/or suboptimal pruning by the indexes [19]. Therefore, workload-driven techniques have been proposed for computing the physical layouts in RDF data management systems [19,20,56,86,107]. These techniques can be evaluated within three categories:

  – techniques for computing the base layout using information in the workload (which is the topic of this thesis),

  – techniques for computing materialized views over one of the workload-oblivious representations [56, 86],

  – techniques for distributing RDF data across multiple compute-nodes using workload information [13, 82, 97, 106, 154, 155].

  Materialized views are widely used in the context of SQL and relational databases [10, 62,92,94,135,137,197,198], and it is only natural to expect similar techniques for RDF databases. To this end, Abadi et al. [4] propose materialized views over the group-by-predicates representation for commonly executed linear queries in the workload. Both Castillo et al. [56] and Goasdoué et al. [86] propose algorithms for computing materialized views over a relational representation of RDF (e.g., single-table representation, group-by-entities, etc.). They also discuss techniques for answering SPARQL queries using these materialized views [86].

  While view materialization can significantly improve performance for a variety of workloads, existing techniques for updating the schemas of the materialized views (and in general, other components of physical design) are scalable only if the changes in the physical schema are minor, which is not always true for SPARQL workloads [30, 124]. In fact, Goasdoué et al. [86] report that one could expect up to 30 minutes for computing these materialized views and they argue that "while this may seem long, [...] the complexity of search is high [...] as view selection is an off-line process.".

Thus, while being workload-driven, existing view materialization techniques over RDF databases are not truly workload-adaptive—one cannot afford to pause query execution for 30 minutes to update the materialized views.

The techniques presented in this thesis go beyond offline tuning by building lightweight indexes that are easy to maintain, through query optimization techniques that allow the underlying base layout to be updated without the need to maintain too much information about the layout, and through partial tuning (i.e., physical design is updated only for small parts of the database).

The third body of work deal with distribution design [57] in RDF databases and advocate for a workload-driven design. This body of work span two types of systems:

- those that rely on MapReduce [69] type of platforms to distribute data and answer queries [154, 155], and
- shared-nothing approaches that rely on a system such as RDF-3x [148] to compute the answers to queries locally within each compute-node, and a master to coordinate and decompose query execution [13, 82, 97, 106].

In both of the above, the "workload-driven" design is concerned with how the RDF graph should be partitioned across the compute-nodes. In systems such as WARP [106] and Partout [82], the graph is partitioned in an offline process. In contrast, systems such as PhDStore [13] support techniques for adaptively shuffling and/or replicating data across the compute-nodes.

While these techniques are important for scaling-out, they are not concerned with the choice of physical layout within each compute-node. In other words, one of the workload-oblivious representations discussed earlier are utilized. For reasons discussed before, this might become a bottleneck for scaling-up. In that respect, the techniques introduced in this thesis are complementary to workload-driven distribution techniques for building systems that can both scale-up and scale-out.

3.1.2 Adaptivity

In this section, the aforementioned techniques are evaluated with respect to adaptivity of physical layout to workload changes. In this regard, systems can be classified into three:

- Systems in which the physical layout is fixed at design and implementation time: In systems like RDF-3x [148], gStore [200, 201], SW-Store [6, 7], the choices made

regarding physical layout and design are hardcoded in the DBMS code, and they cannot change at runtime. Therefore, these systems are not suitable for scenarios in which the types of workloads *cannot* be predicted upfront and/or they fluctuate significantly.

- Systems in which the DBMS gives the flexibility to adjust the physical layout at the time data are loaded and indexed: Systems that rely on an RDF-to-relational mapping [27, 46, 188] fall under this category. However, once the data are loaded, materialized views are generated and indexes are built, it is hard to adapt the physical design of these systems. The problem is that techniques for automatic (physical) schema design [10, 11, 37, 62, 92, 135, 198] and self-tuning databases [52, 60] are not easily applicable to RDF because changes in physical layout in RDF could easily imply switching from a columnar-representation (e.g., group-by-predicates) to a row-oriented representation (e.g., single-table, group-by-entities) which is hard to achieve using existing techniques [19]. The only system that has a hybrid (both row and column-oriented) layout and that can adapt its layout is H_2O [14], but this system is not designed and optimized for RDF data and SPARQL queries.

- Systems that rely on techniques that can automatically and either periodically or continuously adapt their physical design to changing workloads: As mentioned in Chapter 1, while achieving this objective fully remains a vision, the techniques discussed in this thesis provide a strong foundation. In these aspects, the work that is most similar (and related) to the approach taken in this thesis is database cracking [95, 113–115]. Database cracking is a technique implemented in MonetDB [112], where in-memory arrays that represent the columns of the database are clustered (i.e., sorted) and indexed as queries are executed. In other words, each query that is executed on the system is treated as some advice on how each column should be partitioned and sorted. One can think of this as an incremental version of quicksort [65], where the range predicates used in each query are used as the pivots for sorting. Therefore, while the values in each column may initially be in a random order, as queries are executed, the columns get more and more sorted. Furthermore, as the data within each column are partitioned as such, the pivot values are used in building tree-based indexes.

  However, database cracking does not provide a complete solution for this vision because it works only with arrays in the context of in-memory column stores. As discussed earlier, in the context of RDF, column stores are good only for a subset of SPARQL queries. Furthermore, database cracking focuses on indexing; in contrast, data structures and algorithms are needed that support partial tuning for multiple

aspects of physical design in RDF.

3.2 Techniques for Indexing

Indexes used in RDF data management systems can be classified into three categories. The first category consists of conventional B$^+$-tree [36, 64, 126] or hash indexes [77, 126, 131, 132, 136] that are also used in relational database management systems. Most RDF data management systems, including those that rely on the single-table [51, 55, 75, 98, 100, 101, 146, 148, 149, 186, 189] or group-by-entity representations [46, 179, 188], employ this approach. The second category consists of bitmap indexes [151–153]. Bitmap indexes can be used to accompany conventional indexes in columnar RDF stores [6] or in native RDF stores that rely on variations of the group-by-predicates representation [34, 192]. The third category includes graph indexes [48, 120, 182, 190, 191, 194, 196, 199–201].

There are advantages and disadvantages to using indexes from each category. The advantages of using bitmap indexes are that (i) join operations can be implemented efficiently using bitwise operations, and (ii) both the indexes and the intermediate results generated during query evaluation can be represented efficiently in main memory [34]. On the downside, updates can be problematic due to compression/decompression overhead, and due to the fact that large amounts of data need to be moved around [34] (since the bitmap assumes an inherent ordering of tuples which might get broken on updates). In contrast, B$^+$-tree and hash indexes are perhaps easier to update although these updates might still result in random I/O.

The advantage of graph-indexes is that they can exploit the inherent graph-structure of RDF data. For example, gStore [200, 201] creates a signature entry for each vertex in the RDF graph using a variant of Bloom-filters [45] based on the labels of edges that are incident on the vertex and the label of edges that are adjacent to the vertex in question (i.e., 1-hop neighborhood). These signature indexes are organized within a VS*-Tree index [200]. This specialized index has the advantage that star-shaped queries can be evaluated very efficiently [19, 200]. On the other hand, the choice of including incidence/adjacency information only for the 1-hop neighborhood is arbitrary, and is based on *a priori* assumptions about the RDF graph and the workloads. For example, this type of index may not be suitable for graphs with a large number of high-degree vertices because then the VS* may not be effective. Furthermore, this index does not offer any advantage for other types of queries such as linear queries.

A majority of the aforementioned approaches support efficient bulk updates and, to a certain extent, incremental updating of the data at runtime; thus, they are data-adaptive

(i.e., indexes can be updated at runtime based on the changes in the data). On the other hand, these indexing techniques are not workload-adaptive. The problem is that the changes in the workload might dictate a tuning advisor to drop a number of indexes at runtime and create a bunch of new ones, which can be hard to achieve dynamically. (For some systems, this process can take minutes if not hours, even on relatively small datasets.)

Database cracking addresses these problems by embracing the "do not build indexes upfront, but incrementally, as queries are executed" paradigm [95, 113–115]. In database cracking, the sort, partition and merge operations that are commonly used in index construction are deferred until queries are executed. Each query is treated as some advice on how to sort, partition or merge portions of the database. In case of database cracking [95, 113–115], with each query, a portion of an array (in-memory) that corresponds to a column in a column-store gets "cracked" (i.e., sorted, partitioned, merged).

While the indexes used in chameleon-db are inspired by database cracking, there are some differences:

- Database cracking adaptively indexes an array of values from the same domain (i.e., datatype), whereas the Cluster Index in chameleon-db adaptively indexes a set of G-by-Q clusters.
- In database cracking range predicates are used as some advice for building the index, whereas the Cluster Index in chameleon-db relies on the subgraph/supergraph relationships between BGP structures.
- In database cracking, each range query with a single predicate results in at most one additional pivot to be added, whereas in the Cluster Index, the execution of a BGP may result in the addition of more than one pivot to the index.
- In database cracking, the index returns a complete set of answers to the query whereas the Cluster Index may return G-by-Q clusters that are not related to the evaluation of the query (i.e., false positives), which need to be pruned in a validation step during query evaluation. (However, the Cluster Index guarantees that false negatives are not returned).
- In database cracking, total ordering can be imposed on the values that are indexed based on the pivots extracted from the range predicates, whereas it is not possible to impose a total ordering on the G-by-Q clusters based on the pivots extracted from the BGPs.

3.3 Techniques for Query Processing and Optimization

Query evaluation and optimization techniques for SPARQL [26, 84, 90, 94, 110, 147, 156, 160, 168, 171, 176, 195] can be classified into two: (i) those that try to adapt existing query evaluation and optimization strategies in relational databases to RDF and SPARQL [84, 147, 168, 171, 176], and (ii) those that propose solutions for graph-based evaluation and optimization of SPARQL queries [26, 156].

Techniques in the first category consist of (i) equivalence rules for rewriting SPARQL expressions [168], (ii) join reordering techniques based on triple-pattern selectivity estimation [176] and BGP selectivity estimation [147], (iii) adaptation of techniques such as sideways information passing [67, 138, 172] to RDF and SPARQL [147], and (iv) adaptation of query evaluation strategies over materialized views [94] for RDF and SPARQL [86].

Techniques in the second category exploit information about the graph-structure of RDF to achieve further optimization [26]. The query evaluation and optimization strategies presented in this thesis fall into the second category. What makes the techniques proposed in this thesis (Chapter 5) unique is that they assume that the underlying physical layout might be frequently changing, which makes query evaluation harder. The problem is that the query plan generator needs to know about the underlying physical layout—in particular, the way the RDF graph is partitioned to come up with a query plan that produces correct results. On the one hand, one may choose to index and maintain all information about the underlying physical layout, to facilitate query plan generation, but then, updating the underlying physical layout becomes harder because all of the accompanying information needs to be updated as well. On the other hand, one may choose to index and maintain almost nothing about the underlying physical layout, in which case, it is harder to generate query plans, but easier to update the physical layout. Instead, Chapter 5 proposes a solution in-between, where query-rewrite rules are developed that enable efficient generation of query plans while using as little information about the physical layout as possible.

3.4 Techniques for Distribution

There are two main approaches to answering SPARQL queries over very large collections of RDF data [103]. The first approach is *data warehousing*, where data from multiple RDF sources are fetched and consolidated in a data warehouse, and queries are issued against

this data warehouse [103]. The second approach is *query federation*, where SPARQL queries are issued directly on the sources that publish the RDF data, but the results are shipped back to and consolidated at the client-side [103]. There are advantages and disadvantages of each solution [103], but an elaborate discussion is not the topic of this section. The techniques presented in this thesis assume the first approach. Therefore, in the remaining parts, the discussion will be on the *scale-out* techniques for the first approach (i.e., data warehousing).

Broadly speaking, there are three ways of scaling-out data warehousing solutions in RDF. The first (and earliest) body of work rely on Peer-to-Peer (P2P) systems [53,140,169]. The second body of work [63,78,84,109,111,154,155,162,164,166] rely on shared filesystems and/or computing solutions on the cloud [121] such as MapReduce [69], Hadoop [187] and Hadoop Distributed File System (HDFS) [174]. The third body of work [13, 82, 93, 97, 106, 133, 193] rely on shared-nothing architectures [177] where data are distributed across a cluster of machines and each machine is deployed with a system such as RDF-3x [148].

Whether the underlying network model is Peer-to-Peer or client-server or whether the distributed computing model is shared-nothing or shared-all, a common question that all three groups of approaches try to answer is how the RDF database should be partitioned across the nodes in the distributed system. In this respect, the techniques for distributing RDF data across a cluster of machines can be *workload-oblivious* [53, 63, 78, 93, 109, 140] or *workload-driven* [13, 82, 97, 106, 155]. Furthermore, workload-driven techniques can be *offline* [82, 106] or *adaptive* [13, 97, 155].

Workload-oblivious techniques for distribution rely heavily on one or a combination of (i) hash-based partitioning [133], (ii) range-based partitioning [58], or (iii) graph-based partitioning [109]. In hash-based partitioning, RDF data are mapped to compute-nodes based on the outcome of applying a hash function on the values of an attribute (or combinations of multiple attributes) [133]. For example, consider the single-table layout (cf., Figure 1.3a). It is possible to hash-partition this table based on the values of subject attributes (of course, other partitioning schemes are also possible). Range-based partitioning is similar but instead of a hash function, range-predicates are used for partitioning the data [58]. For example, the single-table can be range-partitioned on object values such that objects that are lexicographically within a predefined range are stored in the same compute-node (same goes for other compute-nodes). In graph-based partitioning [109], techniques such as METIS [122] that are based on minimizing graph-cuts can be used to decide which parts of the database should be stored in which compute-nodes.

The problem with any of the aforementioned workload-oblivious techniques is that they are optimal for only certain types of queries. For example, consider hash-partitioning the

single-table representation based on subject values in the table. In this case, triples that share the same subject will be stored in the same compute-node. Therefore, a star-shaped query (i.e., a single join vertex) with all outgoing edges can be answered locally (and in parallel) within each compute-node, and then, the partial results can simply be unioned at the master or the client-side. In contrast, the same statement would not be true for a star query with both incoming and outgoing edges. In that case, query results cannot be computed locally, and in the worst case, they will be computed by joining intermediate results from multiple compute-nodes, which might add complexity [106]. Likewise, in graph-based partitioning, whether or not the query results can be computed locally or by joining intermediate results from multiple partitions depends on how the graph is partitioned and whether a query in the workload coincidentally crosses partition boundaries.

To avoid the aforementioned circumstances in which a query cannot be answered locally within each compute-node, two complementary solutions have been proposed [13, 66, 82, 97, 106, 155]. The first solution is to make more clever decisions about how the RDF data are partitioned by including information about the workload [106]. For example, information about a sample workload can be used as an additional input to the graph-partitioning algorithm [106], so that when optimizing for the cost of a "cut" in the graph, the algorithm also takes into account the potential consequences of the "cut" on evaluating queries in the workload [106]. Likewise, in range-partitioning, it is possible to adjust the range predicates by taking the workload into consideration.

The second solution is to rely on replication [13, 106], where for example, data can still be partitioned using an offline, workload-oblivious technique, but problematic cases (i.e., queries) are handled by maintaining partial copies of the data across compute-nodes so that queries can still be answered locally within each node [13].

Consider the previous hash-partitioning example over the single-table layout, where it was proposed that RDF triples could be hashed based on the subject attribute. It was argued that star-shaped queries with both incoming and outgoing edges could be problematic. A possible workaround to this problem can be to maintain, within each compute-node, an extra set of triples that are just sufficient to answer these problematic queries in the workload [133].

Even though the hierarchical clustering algorithm developed in Section 4.2 has similarities with the workload-driven graph-partitioning algorithms for RDF [13, 82, 97, 106, 155], the following differences need to be highlighted:

- The algorithms for RDF distribution try to partition the RDF graph into as many partitions as there are compute-nodes in the cluster. This is typically in the orders of

thousands. In contrast, the hierarchical clustering algorithm of Section 4.2 generates G-by-Q clusters on the orders of millions.

- Most algorithms for RDF distribution follow a top-down approach (i.e., iteratively divide the RDF graph) whereas the hierarchical clustering algorithm follows a bottom-up approach (i.e., build G-by-Q clusters from smaller graphs).
- While most algorithms for workload-driven RDF distribution try to ensure that subgraphs that match a query in the workload are located in the same partition, the hierarchical clustering places each matching subgraph in a separate G-by-Q cluster.

While a significant step forward, the aforementioned workload-driven techniques may not be sufficient if the workloads change frequently such that there is a need to update the partitioning (or replicated data) on-the-fly. Existing solutions that try to address this problem take one of the following approaches. The first category of solutions dynamically update the replicas that are maintained for each compute-node to ensure that queries can be answered locally even when the workload changes [13]. The second category of solutions implement a caching layer on top of the distributed partitions and try dynamically updating the contents of the cache [155].

Both of these solutions are timely and relevant given the increasing need for supporting diverse and dynamic SPARQL workloads [30,124]. In this respect, the techniques presented in this thesis are also complementary. For example, it might be possible to use TUNABLE-LSH (cf., Section 4.3) to adaptively decide how an RDF graph should be partitioned, what parts of the graph should be replicated and where, and what parts of the graph should be cached.

The techniques presented in this thesis are complementary also for the fact that while adaptive, workload-driven scale-out techniques for RDF [13,97,155] deal with the problem of adaptive distribution design, the RDF engines that are employed within each compute-node use one of the workload-oblivious representations discussed in Chapter 1. In contrast, an ideal solution is one that combines the techniques presented in this thesis for adaptively figuring out the physical layout of RDF within each compute-node with the ones for adaptive distribution design.

3.5 SPARQL Benchmarks

In slightly more than a decade, numerous SPARQL benchmarks have been developed [5, 18, 24, 44, 87, 91, 143, 161, 167]. Earlier benchmarks focused on testing how well systems

implement different use cases of SPARQL [5, 44, 91, 161, 167]. For example, the *Lehigh University Benchmark* (LUBM) [91] was designed for testing the inferencing capabilities of Semantic Web repositories. The *Berlin SPARQL Benchmark* (BSBM) [44] contains multiple use cases such as (i) explore, (ii) update, and (iii) business intelligence use cases. The explore use case is developed around an e-commerce scenario, where queries mimic the search patterns of consumers who are browsing through products. The update use case focuses on testing systems' support for updates, while the business intelligence use case is developed around an OLAP scenario. BSBM also goes into testing how well RDF systems support different (and important) SPARQL features, namely, aggregation, union, and optional graph patterns. The DBpedia SPARQL Benchmark (DBSB) uses real data and real query logs to generate test queries [143].

These earlier benchmarks—even those that rely on real data and real query logs—have been criticized for their lack of diversity, both in their datasets [71] as well as in their query workloads [18]. To address these shortcomings, the Waterloo SPARQL Diversity Test Suite (WatDiv) was developed, which is the benchmark used in the experimental evaluations of this thesis.

Two benchmarking efforts are closely related to WatDiv: (i) RDF benchmarks developed by the Linked Data Benchmark Council (LDBC) [24], and (ii) SPLODGE [87].

LDBC benchmarks are developed in a manual process by a consortium of industrial and academic partners [24]. The process behind the design of LDBC benchmarks is based on identifying "choke-points" [24], i.e., use-cases or queries for which systems are likely to experience problems. In this respect, both LDBC benchmarks and WatDiv are targeting the same problem space. On the other hand, in WatDiv, the process of query template generation is automated, whereas in LDBC, it is manual. Arguably, some of the queries that are generated by WatDiv reflect rare use cases, and relatively few users may be interested in executing such queries, but at the same time, LDBC benchmarks might be missing cases that WatDiv is able to cover.

SPLODGE [87] is a benchmark for federated SPARQL query evaluation. SPLODGE also introduces query features. However, there are three important differences. First, the measures developed in WatDiv are used for evaluating the diversity of queries in benchmarks, whereas the measures introduced in SPLODGE are used in systematically generating diverse queries. To this end, the data-driven measures in WatDiv rely on *actual* selectivity and cardinality values, whereas the ones used in SPLODGE are only *estimates*. Second, selectivity and cardinality measures are defined in a different way in WatDiv. Furthermore, WatDiv is also concerned about the variance in selectivity values, which lets one to consider cases in which all the triple patterns in a BGP contribute evenly to the overall

"selectiveness" of the query versus cases in which only a few triple patterns contribute to the overall "selectiveness" of the query. Third, WatDiv focuses only on BGPs, whereas SPLODGE considers a larger spectrum of SPARQL queries. In these aspects, WatDiv is complementary to SPLODGE and the LDBC benchmarks.

3.6 Locality-Sensitive Hashing (LSH)

Locality-sensitive hashing (LSH) [83,116] has been used in various contexts such as nearest neighbour search [23,31,79,108,116,180], Web document clustering [49,50] and query plan caching [17]. In this paper, we use LSH in the physical design of RDF databases. While multiple families of LSH functions have been developed [50,59,68,83,116], these functions assume that the input distribution is either uniform or static. In contrast, Tunable-LSH can continuously adapt to changes in the input distribution to achieve higher accuracy, which translates to adapting to changes in the query access patterns in the workloads in the context of RDF databases.

Chapter 4

Group-by-Query (G-by-Q) Clustering

This chapter introduces two algorithms for computing group-by-query (G-by-Q) clusters. The first algorithm has a higher computational overhead, but can produce more accurate G-by-Q clusters (Section 4.2). The second algorithm is more dynamic and has a much lower computational overhead, but does so with some loss in accuracy (Section 4.3). Before discussing the details of these two algorithms, it is useful to discuss the objectives of G-by-Q clustering (Section 4.1), which is fundamental to the design of both algorithms.

4.1 Objectives of Group-by-Query (G-by-Q) Clustering

This section introduces a framework for evaluating how good a group-by-query clustering (G-by-Q) is with respect to a workload. A precise definition of group-by-query clustering is as follows:

Definition 6. *Given an RDF graph $G = (V, E)$, a* group-by-query clustering *of G is a set of RDF graphs $\mathbb{P} = \{P_1, \ldots, P_m\}$ such that*

1. *Each P_i is a subgraph of G;*
2. *P_i's are edge disjoint;*
3. *$E(G) = \bigcup_{P_i \in \mathbb{P}} E(P_i)$; and*

4. $V(G) = \bigcup_{P_i \in \mathbb{P}} V(P_i)$. □

In this section, two measures are developed, namely, *segmentation* and *minimality*, that can be utilized for evaluating G-by-Q clusterings. Segmentation is a measure of how distributed the subgraphs that match a BGP are across the group-by-query clusters. Minimality indicates how minimal the clusters are with respect to those subgraphs that match a BGP. It is demonstrated that, often, there is no ideal G-by-Q clustering with respect to a given workload and that the aforementioned measures can be traded-off with one another. Consequently, algorithms that rely on these measures for computing "good" G-by-Q clusterings try to find a balance between the two. Two such algorithms are introduced in this thesis: Section 4.2 presents an algorithm in which the workload is given as input and the algorithm periodically computes a "good" G-by-Q clustering with respect to that workload; and Section 4.3 presents an online algorithm in which G-by-Q clusters are continuously updated to optimize for the most recent queries in the workload.

4.1.1 Overview of Query Evaluation

It was argued in Chapter 1 that group-by-query clustering could improve query performance by (i) reducing physical data fragmentation, (ii) improving the pruning capabilities of the indexes, and (iii) eliminating redundant intermediate result tuples. Some of these optimizations depend on the query evaluation algorithm; therefore, for the sake of completeness, this section provides an overview of query evaluation in chameleon-db. Full details of the algorithm, as well as important design considerations are discussed in Chapter 5.

Evaluating a basic graph pattern over a G-by-Q clustering consists of multiple steps. Let Q denote the basic graph pattern and $\mathbb{P}$ denote the G-by-Q clustering in question. Then, the algorithm operates as follows (the formal algorithm is given in Algorithm 1):

1. First, Q is rewritten as a composition of a set of basic graph patterns, denoted by $\mathbb{Q}_{dec}$ (cf., Algorithm 1, line 3). The details of query decomposition will be discussed in Chapter 5; for now, it is sufficient to assume that an oracle computes the proper decomposition of Q that would produce correct results.

2. For each subquery $Q_i \in \mathbb{Q}_{dec}$, the following steps are taken:

   (a) Any cluster $P_j \in \mathbb{P}$ for which Q_i does not have a matching subgraph (i.e., $[\![Q_i]\!]_{P_j} = \emptyset$), is pruned through an index lookup. The remaining set of clusters are stored in $\mathbb{P}_{rel}$ (cf., Algorithm 1, line 6).

Algorithm 1 Query Evaluation over Group-by-Query (G-by-Q) Clustering

```
1:  procedure EVAL(Q, ℙ)                ▷ Evaluate a BGP Q over a G-by-Q clustering ℙ
2:      R ← ∅                           ▷ Denotes the result of Q
3:      ℚ_dec ← DECOMPOSE(Q)            ▷ Decompose Q into a set of subqueries
4:      for Q_i ∈ ℚ_dec do
5:          R_i ← ∅                     ▷ Denotes the partial result of Q_i
6:          ℙ_rel ← PRUNE(ℙ, Q_i)       ▷ Prune irrelevant clusters
7:          for P_j ∈ ℙ_rel do
8:              R_i ← R_i ∪ [[Q_i]]_{P_j}
9:          end for
10:         R ← R ⋈ R_i
11:     end for
12:     return R
13: end procedure
```

(b) For each cluster in $P_j \in \mathbb{P}_{rel}$, the result of Q_i over P_j is computed (i.e., $[\![Q_i]\!]_{P_j}$).

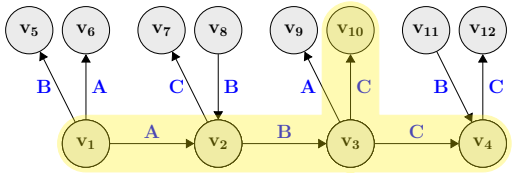
(c) The results from the previous step are unioned, which is stored in R_i.

3. The partial results from Step (2) are joined (cf., Algorithm 1, line 10).

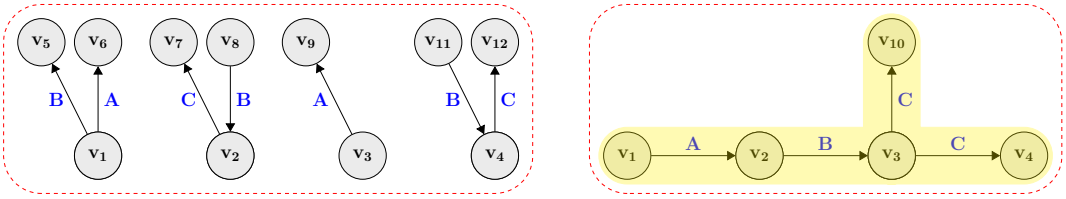
4.1.2 Clustering Objectives

Given a workload, a favorable G-by-Q clustering is one in which (i) RDF triples (or equivalently, edges in the graph representation) that are irrelevant to the evaluation of a query in the workload are separated as much as possible from the relevant ones, and (ii) this property holds for as many queries in the workload as possible. This is explained through an example.

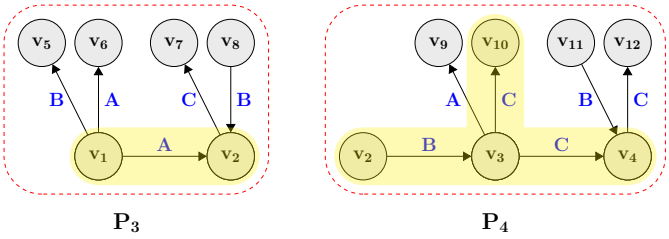
Consider evaluating the linear query $Q = ?w \xrightarrow{A} ?x \xrightarrow{B} ?y \xrightarrow{C} ?z$ against the RDF graph in Figure 4.1a. Any triple that lies outside the colored region is irrelevant to the evaluation of the query. In other words, the result of the query does not change if triples that are outside of the colored region are removed from the RDF graph. Under these circumstances, Clustering A in Figure 4.1b is a better choice for Q than Clustering B in Figure 4.1c. First, all of the triples that are needed for the computation of the query result are contained within a single cluster, namely, P_2. Since the contents of each G-by-Q cluster are also physically clustered in the storage system, the result of the query can be computed more efficiently over Clustering A, that is, with much better I/O and cache utilization.



(a) RDF graph



(b) Clustering A



(c) Clustering B

Figure 4.1: Sample RDF graph and G-by-Q clusterings

	Edge	Source Vertex Degree	Target Vertex Degree	Source Vertex Incoming Edge Signature
P_1	(v_1, A, v_6)	3	1	$\emptyset$
	(v_3, A, v_9)	4	1	$\{B\}$
	(v_1, B, v_5)	3	1	$\emptyset$
	(v_8, B, v_2)	1	4	$\emptyset$
	(v_{11}, B, v_4)	1	3	$\emptyset$
	(v_2, C, v_7)	4	1	$\{A\}$
	(v_4, C, v_{12})	3	1	$\{C\}$
P_2	(v_1, A, v_2)	3	4	$\emptyset$
	(v_2, B, v_3)	4	4	$\{A\}$
	(v_3, C, v_4)	4	3	$\{B\}$
	(v_3, C, v_{10})	4	1	$\{B\}$
	Edge Group	**Source Vertex Max Degree**	**Target Vertex Max Degree**	**Source Vertex Incoming Edge Signature**
P_1	$(*, A, *)$	4	1	$\{B\}$
	$(*, B, *)$	3	4	$\emptyset$
	$(*, C, *)$	4	1	$\{A, C\}$
P_2	$(*, A, *)$	3	4	$\emptyset$
	$(*, B, *)$	4	4	$\{A\}$
	$(*, C, *)$	4	3	$\{B\}$

Table 4.1: An index built over Clustering A in Figure 4.1b

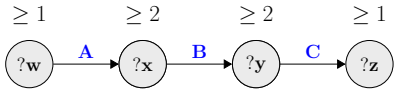


Figure 4.2: Degree and incidence constraints in Q

Second, indexes that are built over Clustering A will be naturally more efficient in pruning clusters as whole: in this case, P_1 does not contain any relevant triples, hence, it can easily be pruned out.

Consider the index in Table 4.1. This index is structured as follows: for each group of edges with the same label in a cluster, the index contains (i) the max degree of the source vertices, (ii) the max degree of the target vertices and (iii) a signature of the labels of the incoming edges on the source vertices. For example, in P_2, two edges have the same label C, namely, (v_3, C, v_4) and (v_3, C, v_{10}). For the edge (v_3, C, v_{10}), v_3 is the source vertex and v_{10} is the target vertex, where the degree of v_3 is 4, the degree of v_{10} is 1, and since (v_2, B, v_3) is the only incoming edge on v_3, the edge signature is $\{B\}$.[1] Consequently, for the edge group $(*, C, *)$ in P_2, where the wildcards can denote any vertex, the max source vertex degree is 4, the max target vertex degree is 3 (which is the degree of v_4), and the incoming edge signature is $\{B\} \cup \{B\} = \{B\}$.

The aforementioned index can be used for pruning P_1 in the evaluation of Q (cf., Figure 4.2). Note that for an edge in the RDF graph in Figure 4.1a to be part of the result of Q, the following conditions must hold:

1. The label of the edge is A, B or C;
2. If the label of the edge is A, its source vertex has degree ≥ 1, and its target vertex has degree ≥ 2;
3. If the label of the edge is B, its source vertex has degree ≥ 2, its target vertex has degree ≥ 2, and the incoming edge signature of its source vertex contains label A; and
4. If the label of the edge is C, its source vertex has degree ≥ 2, its target vertex has degree ≥ 1, and the incoming edge signature of its source vertex contains label B.

[1] Degrees and incident edges are determined over the original RDF graph in Figure 4.1a.

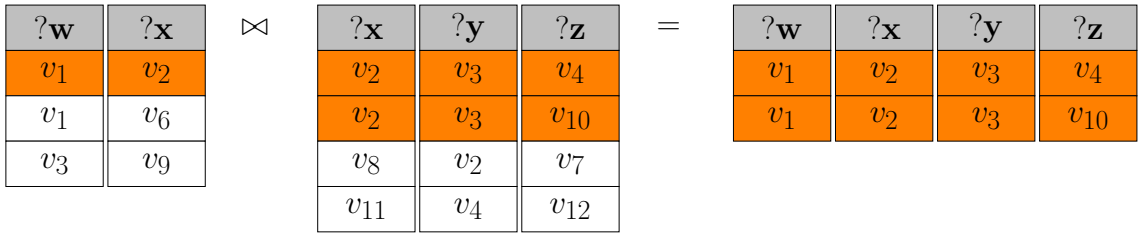


Figure 4.3: Evaluation of Q over Clustering B

These conditions can be easily derived from the structural properties of the query graph, shown in Figure 4.2. Since none of the edges in P_1 satisfy the aforementioned conditions, this cluster can be pruned out.

Third, as a consequence of the pruning of P_1, Q can be evaluated directly over P_2, without decomposition. Consequently, no irrelevant intermediate result tuples are produced, even further improving query evaluation performance.

Next, consider Clustering B in Figure 4.1c, where triples that are irrelevant to the evaluation of Q are mixed with triples that are relevant. In this case, Algorithm 1 would have to decompose Q into at least two subqueries in order to produce the correct query result: $Q_1 = ?w \xrightarrow{A} ?x$ and $Q_2 = ?x \xrightarrow{B} ?y \xrightarrow{C} ?z$. The method of identifying a correct decomposition is discussed in Chapter 5; however, in a nutshell, a basic graph pattern has to be decomposed in Algorithm 1 if any of its matching subgraphs is segmented across multiple clusters. For example, Q has two matching subgraphs: $v_1 \xrightarrow{A} v_2 \xrightarrow{B} v_3 \xrightarrow{C} v_4$ and $v_1 \xrightarrow{A} v_2 \xrightarrow{B} v_3 \xrightarrow{C} v_{10}$; and both of them are segmented across P_3 and P_4 in Clustering B. Then, according to Algorithm 1, each subquery is evaluated over clusters P_3 and P_4, producing two tuple sets, which are joined as shown in Figure 4.3.

In contrast to Clustering A, there are problems with the evaluation of Q over Clustering B. First, triples from which the query result is computed are fragmented across two clusters in the storage system, increasing I/O cost and reducing cache utilization. Second, during query evaluation, it is more difficult to distinguish between relevant and irrelevant triples, which generates unnecessary intermediate result tuples (cf., Figure 4.3). In this case, reordering the join operations or applying sideways information passing [147] to early-prune some of the tuples would not eliminate the problem.

4.1.3 Goodness Measures

Given a basic graph pattern, to quantify how well a group-by-query clustering separates triples that are irrelevant to the evaluation of that BGP from the relevant ones, a combination of two measures is used: *segmentation* and *minimality*. Informally, segmentation is a measure of how distributed the subgraphs that match a BGP are across the group-by-query clusters. Minimality indicates how minimal the clusters are with respect to those subgraphs that match a BGP. These measures are introduced formally in Definition 7.

Definition 7. *Given a clustering $\mathbb{P}$ of an RDF graph G, let*

- Γ_G^Q *denote the set of all distinct subgraphs of G that match a BGP Q, and*
- $E^* = \bigcup_{G' \in \Gamma_G^Q} E(G')$.

Then, segmentation *and* minimality *of $\mathbb{P}$ with respect to Q are defined as follows:*

$$segm_{\mathbb{P}}^{Q} = \left|\left\{(G',P) \in \Gamma_G^Q \times \mathbb{P} \mid E(G') \cap E(P) \neq \emptyset\right\}\right| - \left|\Gamma_G^Q\right|$$

$$minim_{\mathbb{P}}^{Q} = \frac{\left|E^*\right|}{\left|\left\{E(P) \mid P \in \mathbb{P} \text{ and } E(P) \cap E^* \neq \emptyset\right\}\right|}$$ □

The definitions of segmentation and minimality can be easily extended to a query workload $\mathbb{W} = \{Q^1, \ldots, Q^n\}$:

$$segm_{\mathbb{P}}^{\mathbb{W}} = \frac{\sum_{i=1}^{n} segm_{\mathbb{P}}^{Q^i}}{|\mathbb{W}|} \text{ and}$$

$$minim_{\mathbb{P}}^{\mathbb{W}} = \frac{\sum_{i=1}^{n} minim_{\mathbb{P}}^{Q^i}}{|\mathbb{W}|}.$$

Segmentation can take any positive real value, while minimality is always between [0, 1]. An *ideal clustering* for a workload is one whose segmentation is minimal (0) and minimality takes the highest possible value (1). We say that a clustering is *completely segmented* with respect to a query workload if its segmentation is maximal.

Example 5. *Let Q denote the BGP in Figure 4.2. The segmentation of Clustering A in Figure 4.1b with respect to Q is* 0*, while the segmentation of Clustering B in Figure 4.1c is* 2*. These values are computed as follows:*

- *First, note that both Clustering A and Clustering B are G-by-Q clusterings of the RDF graph G in Figure 4.1a. In this case, Γ_G^Q consists of two subgraphs, namely, G_1 and G_2, where*

$$\begin{aligned} G_1 &= (V_1, E_1) & G_2 &= (V_2, E_2) \\ V_1 &= \{v_1, v_2, v_3, v_4\} & V_2 &= \{v_1, v_2, v_3, v_{10}\} \\ E_1 &= \{(v_1, A, v_2), (v_2, B, v_3), & E_2 &= \{(v_1, A, v_2), (v_2, B, v_3), \\ & \qquad (v_3, C, v_4)\} & & \qquad (v_3, C, v_{10})\}. \end{aligned}$$

- *Let $\mathbb{P}_A = \{P_1, P_2\}$ denote Clustering A (cf., Figure 4.1b).*
  - *Then, the Cartesian product $\Gamma_G^Q \times \mathbb{P}_A$ consists of four tuples, that is,*

$$\Gamma_G^Q \times \mathbb{P}_A = \{(G_1, P_1), (G_2, P_1), (G_1, P_2), (G_2, P_2)\}.$$

  - *Among these four tuples, representing pairs of matching subgraphs and G-by-Q clusters, only two pairs have common edges, namely, the pairs (G_1, P_2) and (G_2, P_2).*
  - *Consequently,*

$$\begin{aligned} segm_{\mathbb{P}_A}^Q &= \big|\{(G_1, P_2), (G_2, P_2)\}\big| - \big|\{G_1, G_2\}\big| \\ &= 2 - 2 \\ &= 0. \end{aligned}$$

- *Let $\mathbb{P}_B = \{P_3, P_4\}$ denote Clustering B (cf., Figure 4.1c).*
  - *Then, the Cartesian product $\Gamma_G^Q \times \mathbb{P}_B$ consists of four tuples, that is,*

$$\Gamma_G^Q \times \mathbb{P}_B = \{(G_1, P_3), (G_2, P_3), (G_1, P_4), (G_2, P_4)\}.$$

  - *Among these four tuples, representing pairs of matching subgraphs and G-by-Q clusters, all four pairs have common edges.*
  - *Consequently, $segm_{\mathbb{P}_B}^Q = 4 - 2 = 2$.* □

Example 6. *Let Q denote the BGP in Figure 4.2. The minimality of Clustering A in Figure 4.1b with respect to Q is $\frac{4}{4}$, while the segmentation of Clustering B in Figure 4.1c is $\frac{4}{11}$.*

- *First, note that both Clustering A and Clustering B are G-by-Q clusterings of the RDF graph in Figure 4.1a. Let us call this RDF graph G. In this case, Γ_G^Q consists of two subgraphs, namely, G_1 and G_2, where*

$$\begin{aligned} G_1 &= (V_1, E_1) \\ V_1 &= \{v_1, v_2, v_3, v_4\} \\ E_1 &= \{(v_1, A, v_2), (v_2, B, v_3), \\ &\qquad (v_3, C, v_4)\} \end{aligned} \qquad \begin{aligned} G_2 &= (V_2, E_2) \\ V_2 &= \{v_1, v_2, v_3, v_{10}\} \\ E_2 &= \{(v_1, A, v_2), (v_2, B, v_3), \\ &\qquad (v_3, C, v_{10})\}. \end{aligned}$$

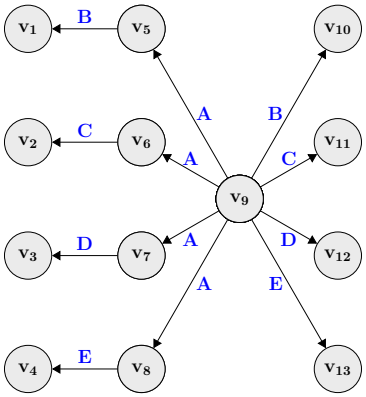
- *Furthermore,*

$$\begin{aligned} E^* &= E(G_1) \cup E(G_2) \\ &= E_1 \cup E_2 \\ &= \{(v_1, A, v_2), (v_2, B, v_3), (v_3, C, v_4), (v_3, C, v_{10})\}. \end{aligned}$$

- *Let $\mathbb{P}_A = \{P_1, P_2\}$ denote Clustering A (cf., Figure 4.1b).*
  - *There is only one cluster in $\mathbb{P}_A$, namely, P_2, such that $E(P_2) \cap E^* \neq \emptyset$.*
  - *Consequently,*

$$minim_{\mathbb{P}_A}^Q = \frac{\left|E^*\right|}{\left|E(P_2)\right|} = \frac{4}{4}.$$

- *Let $\mathbb{P}_B = \{P_3, P_4\}$ denote Clustering B (cf., Figure 4.1c).*
  - *For $i \in \{3, 4\}$, $P_i \in \mathbb{P}_B$ and $E(P_i) \cap E^* \neq \emptyset$.*



(a) Sample RDF graph, G_x (b) Sample BGP, Q_1 (c) Sample BGP, Q_2

Figure 4.4: Sample data and queries for demonstrating the trade-offs between segmentation and minimality

– *Consequently,*

$$\begin{aligned} minim^{Q}_{\mathbb{P}_B} &= \frac{\left|E^*\right|}{\left|E(P_2)\right|} \\ &= \frac{4}{\left|E(P_3) \cup E(P_4)\right|} \\ &= \frac{4}{\left|E(P_3)\right| + \left|E(P_4)\right|} \\ &= \frac{4}{11} \end{aligned}$$

□

4.1.4 Discussion

When there is a single query in the workload, computing the ideal clustering (i.e., whose segmentation = 0 and minimality = 1) is relatively straightforward. On the other hand, when there are multiple queries in the workload, there may not be an ideal clustering at all, let alone a single "best" solution. Often, there is a trade-off between segmentation and minimality, and computing a "good" G-by-Q clustering boils down to tuning these trade-offs.

Consider the RDF graph in Figure 4.4a and the two BGPs in Figure 4.4b and 4.4c. Let

G_x denote the RDF graph and let Q_1 and Q_2 denote the two BGPs, respectively. Q_1 has four matching subgraphs in G_x; more specifically,

$$
\begin{aligned}
&\Gamma_{G_x}^{Q_1} = \{G_x^1, G_x^2, G_x^3, G_x^4\} \text{ such that} \\
&G_x^1 = (V_x^1, E_x^1) \text{ where} && V_x^1 = \{v_1, v_5, v_9\} \\
& && E_x^1 = \{(v_9, A, v_5), (v_5, B, v_1)\} \\
&G_x^2 = (V_x^2, E_x^2) \text{ where} && V_x^2 = \{v_2, v_6, v_9\} \\
& && E_x^2 = \{(v_9, A, v_6), (v_6, C, v_2)\} \\
&G_x^3 = (V_x^3, E_x^3) \text{ where} && V_x^3 = \{v_3, v_7, v_9\} \\
& && E_x^3 = \{(v_9, A, v_7), (v_7, D, v_3)\} \\
&G_x^4 = (V_x^4, E_x^4) \text{ where} && V_x^4 = \{v_4, v_8, v_9\} \\
& && E_x^4 = \{(v_9, A, v_8), (v_8, E, v_4)\} \ ,
\end{aligned}
$$

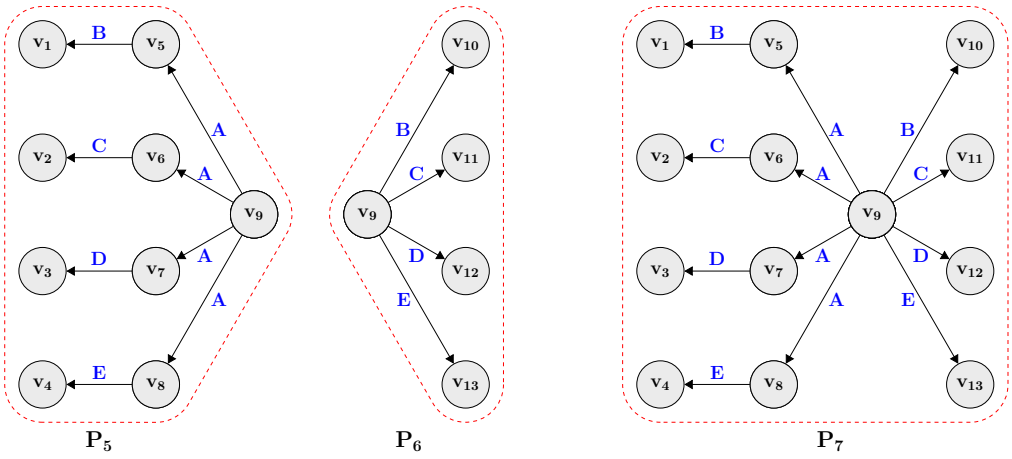
and Q_2 has sixteen matching subgraphs in G_x. Basically, each matching subgraph consists of a pair of edges from the Cartesian product

$$
\begin{aligned}
&\{(v_9, A, v_5), (v_9, A, v_6), (v_9, A, v_7), (v_9, A, v_8)\} \times \\
&\{(v_9, B, v_{10}), (v_9, C, v_{11}), (v_9, D, v_{12}), (v_9, E, v_{13})\}
\end{aligned}
$$ □

Next, consider two possible G-by-Q clusterings of G_x. Let $\mathbb{P}_C = \{P_5, P_6\}$ denote Clustering C in Figure 4.5a, and let $\mathbb{P}_D = \{P_7\}$ denote Clustering D in Figure 4.5b. Assuming that a workload $\mathbb{W}$ consists of Q_1 and Q_2, the corresponding segmentation and minimality values are as follows:

$$
\begin{aligned}
&segm_{\mathbb{P}_C}^{Q_1} = 0 && segm_{\mathbb{P}_D}^{Q_1} = 0 && minim_{\mathbb{P}_C}^{Q_1} = \frac{8}{8} && minim_{\mathbb{P}_D}^{Q_1} = \frac{8}{12} \\
&segm_{\mathbb{P}_C}^{Q_2} = 16 && segm_{\mathbb{P}_D}^{Q_2} = 0 && minim_{\mathbb{P}_C}^{Q_2} = \frac{8}{12} && minim_{\mathbb{P}_D}^{Q_2} = \frac{8}{12} \\
&segm_{\mathbb{P}_C}^{\mathbb{W}} = 8 && segm_{\mathbb{P}_D}^{\mathbb{W}} = 0 && minim_{\mathbb{P}_C}^{\mathbb{W}} = \frac{10}{12} && minim_{\mathbb{P}_D}^{\mathbb{W}} = \frac{8}{12}
\end{aligned}
$$ □

In other words, going from Clustering C to Clustering D, it is possible to reduce segmentation, but that comes at the price of reducing minimality. A low segmentation is desirable, as it generally implies that triples that are relevant to the evaluation of a query in the workload are clustered, but on the other hand, low minimality means that clusters contain triples that are relevant to other queries as well, which may negatively affect performance. These trade-offs are taken into consideration in the next two sections,



(a) Clustering C (b) Clustering D

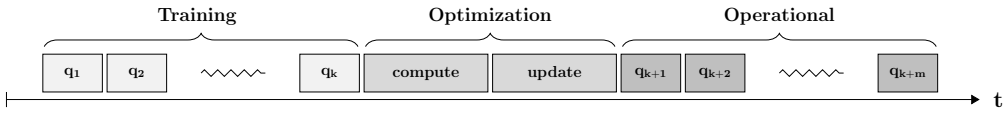
Figure 4.5: Sample G-by-Q clusterings for demonstrating the trade-offs between segmentation and minimality

where two clustering algorithms (periodic and online) are introduced.

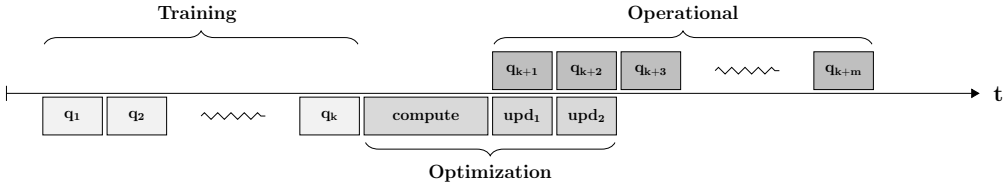
4.2 A Periodic Clustering Algorithm

This section introduces an algorithm for periodically computing group-by-query clusters. The input to the algorithm is a sequence of BGPs (called the *training* workload), which is assumed to be representative of the true workload that will be executed by the system (called the *operational* workload). The algorithm tries to optimize the underlying G-by-Q clustering using the training workload and runs in three phases: (i) training, (ii) optimization, and (iii) operational phases.

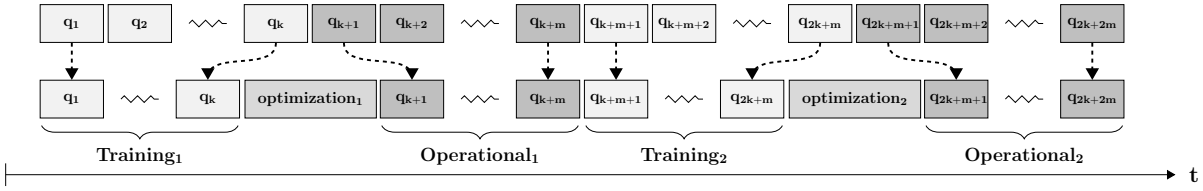
In the first phase, the queries (i.e., BGPs) in the training workload are executed and information is collected. During this phase, the underlying physical representation is *not* updated yet, and thus, queries in the training workload may be executed using a suboptimal clustering. In the optimization phase, information obtained during the training phase is utilized for computing a new G-by-Q clustering that is potentially closer to optimal, and the underlying physical representation is updated based on the computed clusters. In the operational phase, the operational workload is executed on the updated G-by-Q clustering.



(a) Basic Use Case



(b) Interleaved Optimization



(c) The workload is artificially divided into segments of training and operational workloads and the algorithm is repeatedly invoked for each pair of training and operational segments.

Figure 4.6: Use Cases of the Periodic Clustering Algorithm

The aforementioned scenario constitutes the basic use case of the algorithm (cf., Figure 4.6a). In practice, more sophisticated use cases are possible. For example, the optimization and operational phases of the algorithm can be overlapping (cf., Figure 4.6b) as done in chameleon-db [20, 21]. Furthermore, the algorithm can be applied to changing workloads as well, by (i) repeatedly invoking the algorithm at preset intervals or whenever there is detectable change in the characteristics of the workload, and (ii) treating some parts of the workload as training and the remaining parts as operational [20, 21], as demonstrated in Figure 4.6c. This section provides a description of the periodic algorithm assuming the basic use case; more details about the optional uses cases are discussed in Section 4.2.4.

4.2.1 Training Phase

To facilitate the computation of a suitable G-by-Q clustering, in the training phase, upon the execution of a BGP, each distinct subgraph that matches the BGP is annotated with a unique label and a timestamp of query submission. Each annotation is of the form $\langle qid, sid, ts \rangle$, where qid is a unique identifier generated by the system for every BGP that is executed, sid is a unique identifier for each matching subgraph, and ts is a timestamp. Matching subgraphs can be overlapping; therefore, for each edge of a matching subgraph, the annotations are maintained separately. Below are the formal definitions of these concepts.

Definition 8. *An* annotation *is a 3-tuple from the set $\mathcal{A} = \mathbb{Z}^* \times \mathbb{Z}^* \times \mathbb{Z}^*$, where for each annotation $\langle qid, sid, ts \rangle \in \mathcal{A}$,*

- *qid is the query identifier,*
- *sid is the matching subgraph identifier, and*
- *ts is a timestamp.* □

Definition 9. *Given an RDF graph $G = (V, E)$, an* edge annotation *of G is a 2-tuple from the set $\mathcal{M}_G = (E \times \mathcal{A})$.* □

In a given sequence of BGPs, the same query structure may have multiple instantiations. For example, both $Prof_1 \xrightarrow{A} ?x \xrightarrow{B} ?y \xrightarrow{C} VLDB14$ and $Prof_2 \xrightarrow{A} ?x \xrightarrow{B} ?y \xrightarrow{C} SIGMOD14$ are instantiations of the same linear query $?w \xrightarrow{A} ?x \xrightarrow{B} ?y \xrightarrow{C} ?z$. In the training phase, the algorithm detects if a particular query structure occurs frequently, and if so, it tunes

the clustering algorithm to better support instantiations of that query structure. For this purpose, every BGP that is evaluated during the training phase is generalized to its structural form, which means that every URI or literal vertex in the BGP is replaced with a distinct variable as shown above. Then, the frequency of occurrence of each query structure is computed. Finally, the most frequent query structures are executed and the RDF graph is annotated based on their results—just as one would do for any other query in the training workload—to generate annotations also for the generalized BGPs.

The subsequent sections details of the optimization phase, when the clusters are formed.

4.2.2 Design of the Algorithm

To compute the group-by-query clustering, a hierarchical algorithm is used, which starts from a completely segmented clustering, and successively merges clusters until the clustering objective is achieved. The algorithm operates as follows:

S1. Initially, each edge of the RDF graph resides in its own cluster, which corresponds to a completely segmented group-by-query clustering for any possible workload;

S2. The pair of clusters, whose merging improves segmentation (cf., Section 4.1.3) the most, while causing the least trade-off in minimality (cf., Section 4.1.3), is identified (race conditions will be discussed shortly);

S3. Clusters found in Step 2 are merged, which results in a potential decrease in segmentation and/or minimality;

S4. Steps 2–3 are repeated as long as the aggregate minimality of the clustering is greater than a threshold.

It is important to note that segmentation and minimality measures are monotonically decreasing within this algorithm. That is, whenever two clusters are merged, segmentation will potentially decrease because edges with the same *qid* and *sid* labels may be brought together. However, at the same time, edges with different *qid* labels may also be placed in the same cluster, which does not affect segmentation, but reduces minimality.

As discussed in Section 4.1.4, while it is desirable to reduce segmentation, one would also like to improve minimality. That is, when clusters contain too many edges that are individually irrelevant to the execution of the majority of the queries, the overhead of subgraph matching within each cluster can undermine the benefits of reduced segmentation.

In the implementation of chameleon-db, it has been observed that if the clusters contain on average more than 10 times as many irrelevant triples as there are relevant ones, performance of query evaluation starts to degrade. For this reason, the threshold on minimality is set to 0.1 for the experiments with chameleon-db.

There are two reasons for choosing a hierarchical clustering algorithm. First, the way hierarchical clustering works is aligned with the objectives of G-by-Q clustering as clusters are merged one pair at a time until a global objective is achieved. Thus, at each merge step of the algorithm, segmentation can be improved with a quantifiable trade-off in minimality. This would not be so easy to achieve with a centroid-based [118] or spectral [118] clustering algorithm. Second, other algorithms such as k-means [118] require the final number of clusters to be known in advance, which is not possible in the case of G-by-Q clustering.

As an assumption that generally holds, one can expect the final clustering to be fine-grained since subgraphs that match the queries in the workload are likely to be comparable in size to the query graphs, which are relatively small. Furthermore, the final clusters are not likely to be much larger than these subgraphs due to the minimality threshold. Therefore, a bottom-up (agglomerative) approach can reach the clustering objective in fewer number of iterations than a top-down (divisive) approach (hence, the reason why the algorithm starts with a completely segmented clustering and employs an agglomerative approach).

Let $\mathbb{P}$ denote the set of G-by-Q clusters. A critical issue is to decide which pair of clusters to merge in each iteration. In this regard, a distance function $\delta : \mathbb{P} \times \mathbb{P} \rightarrow [0, 1]$ is defined over the clusters such that:

1. $\delta = 1$ is reserved for clusters that should not be merged; and
2. A smaller distance between two clusters implies that the decrease in segmentation is higher (with a lower trade-off in minimality) if these two clusters are merged.

To compute the pairwise distances between clusters, the algorithm relies on the annotations of edges in each pair of clusters, namely, the set of $\langle qid, sid, t\rangle$-tuples. Consequently, the distance between a pair of clusters is defined as a combination of two Jaccard distances: δ_S is defined over the sets of subgraph identifiers (sid), and δ_Q is defined over the sets of query identifiers (qid). For any cluster $P \in \mathbb{P}$, let $\pi_s(P)$ and $\pi_q(P)$ denote the set of subgraph identifiers and the set of query identifiers with which P is annotated, respectively.

Given two clusters P_1 and P_2, the distances $\delta_S(P_1, P_2)$ and $\delta_Q(P_1, P_2)$ are defined as follows:

$$\delta_S = 1 - \frac{|\pi_s(P_1) \cap \pi_s(P_2)|}{|\pi_s(P_1) \cup \pi_s(P_2)|}, \tag{4.1}$$

$$\delta_Q = 1 - \frac{|\pi_q(P_1) \cap \pi_q(P_2)|}{|\pi_q(P_1) \cup \pi_q(P_2)|}. \tag{4.2}$$

The two distance functions are complementary. That is, by merging P_1 with P_2, segmentation decreases by at least $|\pi_s(P_1) \cap \pi_s(P_2)|$, therefore, δ_S is more sensitive to predicting the expected change in segmentation. Likewise, $|\pi_q(P_1) \cup \pi_q(P_2)| - |\pi_q(P_1) \cap \pi_q(P_2)|$ is a more accurate approximation of the expected reduction in minimality; thus, δ_Q is more sensitive to changes in minimality. Hence, the reliance on a combination of both distances. However, in doing so, the algorithm pays particular attention to some race conditions. Specifically, the distance function is designed such that the following order, in which clusters are merged, is always preserved:

R1. A pair of clusters with $\delta_S = 0$ (which also implies that $\delta_Q = 0$) are merged before any other pair of clusters;

R2. Clusters with $\delta_S \neq 0$ and $\delta_Q = 0$, are merged next;

R3. Finally, clusters with $\delta_S \neq 0$ and $\delta_Q \neq 0$ are merged according to a combined distance $\delta = \alpha\delta_S + (1 - \alpha)\delta_Q$, where $\alpha = 0.5$. Choosing α is left as future work.

Note that in the first two cases, minimality will not decrease because the two clusters that are merged have subgraphs that match only a single query. Hence, they are preferred over the third case, in which minimality is expected to decrease. Furthermore, even though the first and second cases are both guaranteed to reduce segmentation (without compromising minimality), the first case can achieve the same objective with smaller clusters, hence, it is preferred over the other. When two clusters P_1 and P_2 are merged, all distances between the new cluster and any other existing cluster P_x for which $\delta(P_1, P_x) < 1$ or $\delta(P_2, P_x) < 1$ need to be updated.

4.2.3 Implementation of the Algorithm

The description of the variables that will be used across the pseudocodes described in this section are given in Table 4.2.

$\mathbb{P}$	The G-by-Q clustering that is being computed
$\mathbf{table_A}$	HASHTABLE $\langle E$, LIST $\langle \mathcal{A} \rangle \rangle$ mapping RDF edges to LIST of annotations
$\mathbf{table_D}$	HASHTABLE mapping RDF graphs to SET of distance structs
levelA, **levelB**	VECTORS of distance structs (used as FIFO queues)
minHeap	Instance of a slightly modified MIN-HEAP data structure

Table 4.2: Description of the common variables used in Algorithms 2–7

Algorithm 2 contains the pseudocode of the clustering algorithm. After initializations of various data structures, the algorithm starts by creating the initial G-by-Q clusters where each triple is placed in its own cluster (lines 4–8), as outlined in S1 in Section 4.2.2. Then, hashtable table_A is populated with the edge annotations M_G that are given as input (lines 9–14). Next, the initial distances between pairs of clusters that share a common vertex are computed and stored in local data structures (lines 15–24) with the help of the UPDATE-DISTANCES function.

Algorithm 2 Cluster

Require:
An RDF graph $G = (V, E)$
A set of edge annotations $M_G = (e, a) \subset \mathcal{M}_G$
context: Variable with pointers to common data structures including
table$_\mathbf{A}$: HASHTABLE $\langle E$, LIST $\langle \mathcal{A} \rangle \rangle$ mapping RDF edges to LIST of annotations

Ensure:
Returns $\mathbb{P}$, a G-by-Q clustering of G based on edge annotations M_G

1: **procedure** CLUSTER(G, M_G, context)
2: $\mathbb{P} \leftarrow \emptyset$
3: $i \leftarrow 0$
4: **for all** $(s, p, o) \in E$ **do**
5: construct $G_i = (V_i, E_i)$ with $V_i = \{s, o\}$ and $E_i = \{(s, p, o)\}$
6: $\mathbb{P} \leftarrow \mathbb{P} \cup G_i$ ▷ (s, p, o) denotes a triple
7: $i \leftarrow i + 1$
8: **end for**
9: **for all** $(e, a) \in M_G$ **do**
10: **if** !context→table$_A$.HASKEY(e) **then**
11: context→table$_A$.INSERT(e, LIST())
12: **end if**
13: context→table$_A$[e].INSERT(a)
14: **end for**
15: **for all** $v \in V$ **do**
16: $E_{inc} \leftarrow$ INCIDENT(G, v) ▷ INCIDENT finds edges incident on v in G
17: **for all** $e_A \in E_{inc}$ **do**
18: **for all** $e_B \in E_{inc} \setminus \{e_A\}$ **do**
19: $G_A \leftarrow$ CONTAINER($\mathbb{P}$, e_A) ▷ CONTAINER finds cluster in $\mathbb{P}$
20: $G_B \leftarrow$ CONTAINER($\mathbb{P}$, e_B) ▷ that contains a given edge
21: UPDATE-DISTANCES(G_A, G_B, context)
22: **end for**
23: **end for**
24: **end for**
25: minHeap.BUILD()
26: **while** !levelA.EMPTY() **do**
27: d ← levelA.POP-FRONT()
28: MERGE(d.first, d.second, $\mathbb{P}$, context) ▷ Clusters that satisfy R1 are merged
29: **end while**

```
30:     while !levelB.Empty() do
31:         d ← levelB.Pop-Front()
32:         Merge(d.first, d.second, ℙ, context)      ▷ Clusters that satisfy R2 are merged
33:     end while
34:     while !minHeap.Empty() and Minimality(ℙ) ≥ Θ do
35:         d ← minHeap.Delete()
36:         Merge(d.first, d.second, ℙ, context)      ▷ Clusters that satisfy R3 are merged
37:     end while
38:     return ℙ
39: end procedure
```

The pseudocode of the Update-Distances function is given in Algorithm 3. In line 8, the function calls Construct-Distances (Algorithm 4) to compute distances δ_S, δ_Q and δ between a pair of RDF graphs G_i and G_j. As outlined in R1–R3 in Section 4.2.2, distances are categorized into three groups (i.e., to avoid race conditions), and stored respectively in three variables: levelA, levelB and minHeap (lines 9–25). The variables levelA and levelB are FIFO queues and minHeap is a slightly modified minimum heap (binary) that can accommodate updates given the index of an element within the heap. For simplicity, it is also assumed that the Reverse-Lookup function of minHeap returns the index of a given element. Irrespective of its category, the distance between two RDF graphs G_i and G_j are stored in hashtable$_D$ with keys G_i and G_j, respectively, to be used for fast lookups in other parts of the algorithm.

Once initial distances are computed, Algorithm 2 builds the minimum heap (line 25). In the last phase, clusters are merged a pair at a time with the help of the Merge function (Algorithm 6), while ensuring that the race conditions are respected (cf., R1–R3 in Section 4.2.2) and minimality threshold is not reached (lines 26–38).

The Merge function (Algorithm 6) operates as follows. Given a pair of RDF graphs G_1 and G_2, first, G_M is constructed by merging G_1 and G_2 (line 2). Then, old distances are updated with the new distances between the merged RDF graph G_M and

- all the neighbors of G_1 (i.e., those with distances less than 1) except G_2, and
- all the neighbors of G_2 (i.e., those with distances less than 1) except G_1 (lines 3–4).

For this purpose, the helper function Traverse Neighbors (cf., Algorithm 7) is invoked. Next, the old distances are removed from hashtable table$D$ (lines 5–8). Lastly, $\mathbb{P}$, which represents the G-by-Q clustering is updated (line 9).

Due to the minimality threshold, it is safe to assume that the CONSTRUCT-DISTANCES procedure (cf., Algorithm 4) always operates on a pair of G-by-Q clusters that are relatively small (i.e., with respect to the entire RDF graph), hence, the procedure's computational complexity is $O(1)$. Therefore, the computational complexity of the UPDATE-DISTANCES procedure (cf., Algorithm 3) is determined by the cost of updating the minimum heap, which is $O\big(log(n)\big)$, where n represents the number of elements stored in the minimum heap. Based on lines 15–25 in Algorithm 2, it is safe to assume that $n \approx O\big(\frac{|E|^2}{|V|}\big)$, where $|E|$ denotes the number of edges in the RDF graph and $|V|$ denotes the number of vertices. Consequently, lines 2–25 of Algorithm 2 have $O\big(n\, log(n)\big)$ computational complexity.

Estimating the computational complexity of the TRAVERSE-NEIGHBORS procedure (cf., Algorithm 7) is not as trivial: In the worst case, all the other G-by-Q clusters need to be considered for each invocation of TRAVERSE-NEIGHBORS, which can be $O(n)$ in number. On the other hand, in practice, this number is much smaller due to the minimality threshold. Assuming the worst-case, TRAVERSE-NEIGHBORS has $O\big(n\, log(n)\big)$ complexity because UPDATE-DISTANCES is invoked $O(n)$ times. Therefore, lines 26–38 of Algorithm 2 have $O\big(n^2\, log(n)\big)$ worst-case complexity, which is also the overall worst-case complexity of Algorithm 2. A closer look at Algorithm 2 suggests that it is an *average-linkage* clustering algorithm [139], thus, verifying the complexity analysis.

In practice, Algorithm 2 rarely hits its worst-case complexity because of two reasons. First, only a small portion of the RDF graph is annotated, thus, reducing the number of G-by-Q clusters that can be merged. Second, due to the minimality threshold, TRAVERSE-NEIGHBORS rarely considers $O(n)$ clusters. While an average-case complexity analysis is beyond the scope of this thesis, experimental evaluation of Algorithm 2 suggests that the computational overhead for the workloads considered is less than a second (cf., Section 6.2.5). Nevertheless, Section 4.3 introduces a much more predictable algorithm with $O(|E|)$ worst-case computational complexity.

4.2.4 Updating the Physical Representation

Once a suitable G-by-Q clustering is computed, the system performs the transformation from the current physical representation to the desired one as a set of atomic update operations (i.e., deletion and insertion) on the set of physical clusters in the storage system. Each operation has the property that before and after the operation, the database represents exactly the same RDF graph but using a different clustering.

The update operations are executed concurrently with the queries, which is possible because the query evaluation approach introduced in Section 4.1.1 is designed with an

isolation guarantee: once results are computed within a cluster, query evaluation does not need to access that cluster anymore, allowing it to be updated while query evaluation proceeds over other clusters. In order to ensure that updates do not take place before a query has completely "consumed" the contents of a cluster, a two-level locking scheme is used.

More specifically, two types of locks are maintained: (i) a *global lock* on the database, and (ii) a *cluster-based lock* on each G-by-Q cluster. A query needs to obtain and hold on to the global lock until it fully determines which clusters are relevant to its execution (recall that using the indexes it is possible to prune out irrelevant clusters, cf., Algorithm 1). At this time, no updates can take place as they are also requesting the global lock. Then, the query (i) obtains a cluster-based lock on each cluster that is relevant to its execution, (ii) releases the global lock, and (iii) starts computing results within each cluster. The isolation guarantee of Algorithm 1 plays an important role here because queries can be evaluated independently of each cluster, and the results from one cluster do not interfere with another. Therefore, as soon as the results are computed within a cluster, the query can release the cluster-based lock, allowing that cluster to be updated. Now, an update operation can successfully obtain, first the global lock and then the cluster-based lock, and carry out its task. When the operation is complete, it releases both locks. The aforementioned locking scheme can be easily extended to support concurrent execution of queries (which is a topic of future work), in which case, (read-only) queries can obtain *shared* cluster-based locks while the update operations need to obtain *exclusive* locks.

4.2.5 Discussion

The aforementioned mechanism enables complex use cases such as the ones depicted in Figures 4.6b and 4.6c to be implemented, and to a certain extent, enables the algorithm to be used for changing workloads as well (i.e., through repeated invocation of the algorithm). However, the next section presents a different solution that can adapt even more easily to changing workloads.

Algorithm 3 Update Distances

Require:

$\mathbf{G_i}$, **$\mathbf{G_j}$**: Pair of RDF graphs between which distances are computed

context: Variable with pointers to common data structures including

$\mathbf{table_A}$: HASHTABLE $\langle E$, LIST $\langle \mathcal{A} \rangle \rangle$ mapping RDF edges to LIST of annotations

$\mathbf{table_D}$: HASHTABLE mapping RDF graphs to SET of distance structs

levelA, **levelB**: VECTORS of distance structs

minHeap: Instance of a slightly modified MIN-HEAP data structure

Ensure:

Distances δ_S, δ_Q and δ between two annotated RDF graphs G_i, G_j are computed, and relevant data structures are updated

```
1: procedure UPDATE-DISTANCES(G_i, G_j, context)
2:     if !context→table_D.HASKEY(G_i) then
3:         context→table_D.INSERT(G_i, LIST())
4:     end if
5:     if !context→table_D.HASKEY(G_j) then
6:         context→table_D.INSERT(G_j, LIST())
7:     end if
8:     ⟨ d_S, d_Q, d ⟩ ← CONSTRUCT-DISTANCES(G_i, G_j, context→table_A)
9:     if d_S.distance = 0 then
10:        context→levelA.PUSH-BACK(d_S)
11:        context→table_D[G_i].INSERT(d_S)
12:        context→table_D[G_j].INSERT(d_S)
13:    else if d_Q.distance = 0 then
14:        context→levelB.PUSH-BACK(d_Q)
15:        context→table_D[G_i].INSERT(d_Q)
16:        context→table_D[G_j].INSERT(d_Q)
17:    else if d.distance < 1 then
18:        if (id ← minHeap.INV-LOOKUP(d)) > 0 then
19:            context→minHeap.UPDATE(id, d)
20:        else
21:            context→minHeap.INSERT(d)
22:        end if
23:        context→table_D[G_i].INSERT(d)
24:        context→table_D[G_j].INSERT(d)
25:    end if
26: end procedure
```

Algorithm 4 Construct Distances

Require:
Two RDF graphs $G_i = (V_i, E_i)$ and $G_j = (V_j, E_j)$ over which distances are to be computed
table$_\mathbf{A}$: HASHTABLE $\langle E$, LIST $\langle \mathcal{A} \rangle \rangle$ mapping RDF edges to LIST of annotations
Ensure:
Computes and returns a 3-tuple consisting of structs containing d_S, d_Q and d distances
1: **procedure** CONSTRUCT-DISTANCES(G_i, G_j, table$_A$)
2: $\Pi_i \leftarrow$ GET-ANNOTATIONS(G_i, table$_A$)
3: $\Pi_j \leftarrow$ GET-ANNOTATIONS(G_j, table$_A$)
4: $\delta_S \leftarrow$ COMPUTE-S-DISTANCE(Π_i, Π_j) ▷ cf., Equation 4.1
5: $\delta_Q \leftarrow$ COMPUTE-Q-DISTANCE(Π_i, Π_j)
6: $\delta \leftarrow (\delta_S + \delta_Q)/2$
7: DistanceStruct $\mathrm{d}_S(G_i, G_j, \delta_S)$
8: DistanceStruct $\mathrm{d}_Q(G_i, G_j, \delta_Q)$
9: DistanceStruct $\mathrm{d}(G_i, G_j, \delta)$
10: **return** $\langle \mathrm{d}_S, \mathrm{d}_Q, \mathrm{d} \rangle$
11: **end procedure**

Algorithm 5 Get Annotations

Require:
$\mathbf{G} = (\mathbf{V}, \mathbf{E})$: An RDF graph
table$_\mathbf{A}$: HASHTABLE $\langle E$, LIST $\langle \mathcal{A} \rangle \rangle$ mapping RDF edges to LIST of annotations
Ensure:
Π, the union of annotations for all edges in G is returned
1: **procedure** GET-ANNOTATIONS(G, table$_A$)
2: $\Pi \leftarrow \emptyset$
3: **for all** $e \in E$ **do**
4: **if** table$_A$.HASKEY(e) **then**
5: **for all** $a \in$ table$_A[e]$ **do**
6: $\Pi \leftarrow \Pi \cup a$
7: **end for**
8: **end if**
9: **end for**
10: **return** Π
11: **end procedure**

Algorithm 6 Merge

Require:

$\mathbf{G_1}$, **$\mathbf{G_2}$**: Two RDF graphs that are being merged

$\mathbb{P}$: The G-by-Q clustering to be updated

context: Variable with pointers to common data structures including

$\mathbf{table_D}$: HASHTABLE mapping RDF graphs to SET of distance structs

Ensure:

Merge RDF graphs G_1 and G_2

1: **procedure** MERGE(G_1, G_2, $\mathbb{P}$, context)
2: construct $G_M = (V_M, E_M)$ with $V_M = V_1 \cup V_2$ and $E_M = E_1 \cup E_2$
3: garbage $\leftarrow$ TRAVERSE-NEIGHBORS(G_1, G_2, G_M, context)
4: garbage $\leftarrow$ garbage $\cup$ TRAVERSE-NEIGHBORS(G_2, G_1, G_M, context)
5: **for all** $\mathrm{d}_{old} \in$ garbage **do**
6: context$\rightarrow$table$_D$[d_{old}.first].REMOVE(d_{old})
7: context$\rightarrow$table$_D$[d_{old}.second].REMOVE(d_{old})
8: **end for**
9: $\mathbb{P} \leftarrow (\mathbb{P} \setminus \{G_1, G_2\}) \cup G_M$
10: **end procedure**

Algorithm 7 Traverse Neighbors

Require:

G_i: RDF graph whose neighbors are being traversed

G_j: RDF graph to be excluded from traversal because it is being merged with G_i

G_M: RDF graph resulting from the merge of G_i and G_j

context: Variable with pointers to common data structures including

table$_D$: HASHTABLE mapping RDF graphs to SET of distance structs

Ensure:

Neighbors of G_i are traversed (excluding G_j), distances between each neighbor of G_i and the newly merged graph G_M are computed, and relevant data structures are updated; Stale distance structs are collected and returned for garbage collection.

```
1:  procedure TRAVERSE-NEIGHBORS(G_i, G_j, G_M, context)
2:      garbage ← ∅
3:      for all d_old ∈ context→table_D[G_i] do
4:          if d_old.first= G_i AND d_old.second≠ G_j then
5:              G_N ← d_old.second
6:          else if d_old.first≠ G_j AND d_old.second= G_i then
7:              G_N ← d_old.first
8:          else
9:              continue
10:         end if
11:         UPDATE-DISTANCES(G_M, G_N, context)
12:         garbage ← garbage ∪ d_old
13:     end for
14:     return garbage
15: end procedure
```

4.3 An Online Clustering Algorithm

Whenever a SPARQL query is executed, there is an opportunity to observe how records in an RDF database are being utilized. This information about query access patterns can be used to dynamically compute the G-by-Q clusters in the storage system. Dynamism is important in RDF systems because of the high variability and dynamism in SPARQL workloads [30, 124]. While this problem has been studied as physical clustering [135] and distribution design [57], the highly dynamic nature of the queries over RDF data introduces new challenges. First, traditional algorithms are offline, and since clustering is

an NP-hard problem and most approximations have quadratic complexity [118], they are not suitable for online database clustering. Instead, techniques are needed with similar clustering objectives, but that have constant running time. Second, systems are typically expected to execute most queries in subseconds [145], leaving only fractions of a second to update their physical data structures (i.e., in case of chameleon-db, the main concern is dynamically moving RDF triples across the storage system).

This section addresses the aforementioned issues by making two contributions. First, as shown in Figure 1.7, instead of clustering the whole database, only the "hot" portions of the database are clustered by relying on the admission policy of the existing database cache. Second, a self-tuning locality-sensitive hash (LSH) function, namely, Tunable-LSH is developed to decide in constant-time where in the storage system to place a triple. Tunable-LSH has two important properties:

- It tries to ensure that (i) triples with *similar* utilization patterns (i.e., those triples that are co-accessed across similar sets of queries) are mapped as much as possible to the same G-by-Q clusters (hence, pages in the storage system) while (ii) minimizing the number of triples with *dissimilar* utilization patterns that are falsely mapped to the same G-by-Q cluster.
- Unlike conventional LSH [83,116], Tunable-LSH can auto-tune so as to achieve the aforementioned clustering objectives with high accuracy even when the workloads change.

In determining which RDF triples are accessed together, the techniques in this section rely on the labeling scheme introduced in Section 4.2.1. That is, whenever a BGP is executed, each matching subgraph in the result of the BGP is marked with a unique identifier (i.e., *sid*). Then, Tunable-LSH is used to cluster those RDF triples that share the same identifier. The aforementioned subproperty (i) of Tunable-LSH ensures that the *segmentation* of the generated G-by-Q clustering is low, while subproperty (ii) ensures that its *minimality* is high (cf., Section 4.1).

These ideas are illustrated in Figure 1.7. Let us assume that initially, the triples are *not* clustered according to any particular workload. Therefore, the performance of the system is suboptimal. However, every time triples are fetched from the storage system, there is an opportunity to bring together into the same G-by-Q cluster those triples that are co-accessed but are fragmented across the storage system. Tunable-LSH achieves these with minimal overhead. Furthermore, Tunable-LSH is continuously updated to reflect any changes in the workload characteristics. Consequently, as more queries are executed, triples in the database become more clustered, and the performance of the system improves.

The rest of this section is organized as follows: Section 4.3.1 gives a conceptual description of the problem. Section 4.3.2 describes the overview of our approach while the remaining sections provide the details.

4.3.1 Preliminaries

In this section, the term "record" is used to denote a single RDF triple. However, the following conceptualization can be generalized to other physical layouts, where a record contains multiple RDF triples.

Given a sequence of database records that represent their serialization order in the storage system, the access patterns of a query can conceptually be represented as a bit vector, where a bit is set to 1 if the corresponding record in the sequence is accessed by the query. This bit vector is called a *query access vector* ($\vec{q}$).

As more queries are executed, their query access vectors can be accumulated column-by-column in a matrix, as shown in Figure 4.7a. This matrix is called a *query access matrix*. For presentation, it is assumed that queries are numbered according to their order of execution by the RDF data management system. In chameleon-db, for each matching subgraph of a BGP, a separate column is allocated in the matrix. In practice, other groupings are also possible.

Each row of the query access matrix constitutes what is called a *record utilization vector* ($\vec{r}$), which represents the set of queries that access record r. As a convention, to distinguish between a query and its access vector (likewise, a record and its utilization vector), the symbols q and $\vec{q}$ (likewise, r and $\vec{r}$) are used, respectively. The complete list of symbols that are used in this section are given in Table 4.3.

To model the memory hierarchy, an additional notation is used in the matrix representation: records that are physically stored together on the same disk/memory page are grouped together in the query access matrix. For example, Figure 4.7a and Figure 4.7b represent two alternative ways in which the records in an RDF database can be clustered (groups are separated by horizontal dashed lines). Even though both figures depict essentially the same query access patterns, the physical organization in Figure 4.7b is preferable, because in Figure 4.7a, most queries require access to 4 pages each, whereas in Figure 4.7b, the number of accesses is reduced by almost half.

Given a sequence of queries and the number of pages in the storage system, the objective of the online clustering algorithm is to *compute a G-by-Q clustering such that records with similar utilization vectors are grouped together so as to minimize the total number of page*

$$
\begin{array}{c} \\ r_0 \\ r_1 \\ \\ r_2 \\ r_3 \\ \\ r_4 \\ r_5 \\ \\ r_6 \\ r_7 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 \end{array} \\
\left(\begin{array}{cccc:cccc}
0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\
\mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} \\
\mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 \\
\hline
\mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 \\
0 & \mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} \\
\hline
\mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right)
\end{array}
$$

(a) Representation at $t = 8$.

$$
\begin{array}{c} \\ r_7 \\ r_1 \\ \\ r_4 \\ r_3 \\ \\ r_6 \\ r_0 \\ \\ r_5 \\ r_2 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 \end{array} \\
\left(\begin{array}{cccc:cccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\
\hline
\mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 \\
\mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 \\
\hline
\mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\
0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\
\hline
0 & \mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1} \\
0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & \mathbf{1}
\end{array}\right)
\end{array}
$$

(b) Clustered on rows

$$
\begin{array}{c} \\ r_7 \\ r_1 \\ \\ r_4 \\ r_3 \\ \\ r_6 \\ r_0 \\ \\ r_5 \\ r_2 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} q_0 & q_2 & q_6 & q_4 & q_3 & q_5 & q_7 & q_1 \end{array} \\
\left(\begin{array}{cccc:cccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
\mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\
\mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & 0 \\
\hline
\mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\
0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\
\hline
0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\
0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0
\end{array}\right)
\end{array}
$$

(c) Clustered on rows and columns

$$
\begin{array}{c} \\ c_7 \\ c_1 \\ \\ c_4 \\ c_3 \\ \\ c_6 \\ c_0 \\ \\ c_5 \\ c_2 \end{array}
\begin{array}{c}
\begin{array}{cc} G_0 & G_1 \end{array} \\
\left(\begin{array}{cc}
0 & 0 \\
2 & 0 \\
\hline
2 & 2 \\
2 & 2 \\
\hline
4 & 4 \\
3 & 4 \\
\hline
2 & 2 \\
1 & 2
\end{array}\right)
\end{array}
$$

(d) Grouping of bits

$$
\begin{array}{c} \\ c_7 \\ c_1 \\ \\ c_4 \\ c_3 \\ \\ c_6 \\ c_0 \\ \\ c_5 \\ c_2 \end{array}
\begin{array}{c}
\begin{array}{cc} G_0 & G_1 \end{array} \\
\left(\begin{array}{cc}
0 & 0 \\
2 & 0 \\
\hline
4 & 0 \\
4 & 0 \\
\hline
4 & 4 \\
3 & 4 \\
\hline
0 & 4 \\
0 & 3
\end{array}\right)
\end{array}
$$

(e) Alternative grouping

Figure 4.7: Matrix representation of query access patterns.

accesses. To determine the similarity between record utilization vectors, the following property is used. Two records are co-accessed by a query if both of the corresponding bits in that query's access vector are set to 1. Extending this concept to a set of queries, it is assumed that two records are co-accessed across multiple queries if the corresponding bits

	Symbol	*Description*
Constants	ω	database size (i.e., number of records)
	ϵ	number of pages in the storage system
	k	maximum no. of query access vectors that can be stored
	b	number of entries in each record utilization counter
	t	current time
Data structures	$\vec{q}$	query access vector (contains ω bits)
	$\vec{r}$	record utilization vector (contains k bits)
	$\vec{c}$	record utilization counter (contains b entries)
	$\vec{P}$	depending on the context, a point in a k-dimensional or b-dimensional (Taxicab) space
	$M_{\omega \times k}$	query access matrix; contains the last k most representative query access vectors (in columns), or equivalently, ω record utilization vectors (in rows)
	$C_{\omega \times b}$	frequency matrix; represents record utilization frequency over b groups of query access vectors
Accessors	$q[i]$	value of the i^{th} bit in query access vector $\vec{q}$
	$r[i]$	value of the i^{th} bit in record utilization vector $\vec{r}$
	$c[i]$	value of the i^{th} entry in record utilization counter $\vec{c}$
	$P[i]$	value of the i^{th} coordinate in point $\vec{P}$
	$M[i][j]$	value of the i^{th} row and j^{th} column in matrix
	$C[i][j]$	value of the i^{th} row and j^{th} column in matrix
Distances	$\delta(\vec{r_x}, \vec{r_y})$	Hamming distance between two record utilization vectors
	$\delta^H(\vec{q_x}, \vec{q_y})$	MIN-HASH distance between two query access vectors
	$\delta^M(\vec{P_x}, \vec{P_y})$	Manhattan distance between two points

Table 4.3: Symbols used throughout Section 4.3

in the record utilization vectors are set to 1 for all the queries in the set. For example, according to Figure 4.7a, records r_1 and r_3 are co-accessed by queries q_0 and q_2, and records r_0 and r_6 are co-accessed across the queries q_1–q_7.

Given a sequence of queries, it may often be the case that a pair of records are not co-accessed in *all* of the queries. Therefore, to measure the extent to which a pair of records are co-accessed, their Hamming distance [96] is used. Specifically, given two record utilization vectors for the same sequence of queries, their Hamming distance—denoted as

$\delta(\vec{q}_x, \vec{q}_y)$—is defined as the minimum number of substitutions necessary to make the two bit vectors the same [96].[2] Hence, the smaller the Hamming distance between a pair of records, the greater the extent to which they are co-accessed.

Consider the record utilization vectors $\vec{r}_0$, $\vec{r}_2$, $\vec{r}_5$ and $\vec{r}_6$ across the query sequence q_0–q_7 in Figure 4.7a. The pairwise Hamming distances are as follows: $\delta(r_0, r_6) = 1$, $\delta(r_2, r_5) = 1$, $\delta(r_0, r_5) = 3$, $\delta(r_0, r_2) = 4$, $\delta(r_5, r_6) = 4$ and $\delta(r_2, r_6) = 5$. Consequently, to achieve better physical clustering, the online clustering algorithm should try to store r_0 and r_6 together and r_2 and r_5 together, while keeping r_0 and r_6 apart from r_2 and r_5.

4.3.2 Overview of Tunable-LSH

The dynamic nature of queries over RDF data necessitate a solution different than existing clustering algorithms [19]. That is, while conventional clustering algorithms [118] might be perfectly applicable for the *offline* tuning of a database, in an *online* scenario, what is needed is an algorithm that clusters records on-the-fly and within microseconds. Clustering is an NP-complete problem [118], and most approximations take at least quadratic time. It is not very well-understood which clustering algorithm is more suitable for which types of input distributions [8], let alone the fact that incremental versions of these algorithms are largely domain-specific [9]. In contrast, Tunable-LSH is a self-tuning locality-sensitive hash (LSH) function, which is used as follows:

As records are fetched from the storage system, records that are fragmented are identified. Then, Tunable-LSH is used to decide, in constant-time, how a fragmented record needs to be clustered into G-by-Q clusters in the storage system (cf., Figure. 1.7). Furthermore, methods are developed to continuously auto-tune this LSH function to adapt to changing query access patterns that are encountered while executing the workload. This way, Tunable-LSH can achieve much higher clustering accuracy than conventional LSH schemes, which are static.

Let $\mathbb{Z}_{\alpha \dots \beta}$ denote the set of integers in the interval $[\alpha, \beta]$, and let $\mathbb{Z}^n_{\alpha \dots \beta}$ denote the n-fold Cartesian product:

$$\underbrace{\mathbb{Z}_{\alpha \dots \beta} \times \cdots \times \mathbb{Z}_{\alpha \dots \beta}}_{n}.$$

[2] The Hamming distance between two record utilization vectors is equal to their edit distance [134], as well as the Manhattan distance [128] between these two vectors in l_1 norm.

Furthermore, assume that a non-injective, surjective function $f : \mathbb{Z}_{0\,\cdots\,(k-1)} \rightarrow \mathbb{Z}_{0\,\cdots\,(b-1)}$ is given, where $b \ll k$, and for all $y \in \mathbb{Z}_{0\,\cdots\,(b-1)}$, it holds that

$$\Big|\{x : f(x) = y\}\Big| \leq \left\lceil \frac{k}{b} \right\rceil.$$

In other words, f is a hash function with the property that, given k input values and b possible outcomes, no more than $\lceil \frac{k}{b} \rceil$ values in the domain of the function will be hashed to the same value. Section 4.3.4 discusses in more detail how f can be constructed. Then, TUNABLE-LSH is defined as $h : \mathbb{Z}^k_{0\,\cdots\,1} \rightarrow \mathbb{Z}_{0\,\cdots\,(\epsilon-1)}$, where ϵ represents the number of pages in the storage system. More specifically, h is defined as a composition of two functions h_1 and h_2.

Definition 10 (TUNABLE-LSH)**.**

Let

$$\vec{r} = (r[0], \ldots, r[k-1]) \in \mathbb{Z}^k_{0\,\cdots\,1}, \text{ and}$$
$$\vec{c} = (c[0], \ldots, c[b-1]) \in \mathbb{Z}^b_{0\,\cdots\,\lceil \frac{k}{b} \rceil}.$$

Then, a tunable LSH function h is defined as

$$h = \mathbf{h_2} \circ \mathbf{h_1}$$

where

$\mathbf{h_1} : \mathbb{Z}^k_{0\,\cdots\,1} \rightarrow \mathbb{Z}^b_{0\,\cdots\,\lceil \frac{k}{b} \rceil}$*, where* $h_1(\vec{r}) = \vec{c}$ *iff*

$$\forall y \; c[y] = \sum_{x=0}^{k-1} \begin{cases} r[x] & : f(x) = y \\ 0 & : f(x) \neq y \end{cases}$$

$\mathbf{h_2} : \mathbb{Z}^b_{0\,\cdots\,\lceil \frac{k}{b} \rceil} \rightarrow \mathbb{Z}_{0\,\cdots\,(\epsilon-1)}$*, where* $h_2(\vec{c}) = v$ *and* v *is the coordinate of* $\vec{c}$ *(rounded up to the nearest integer) on a space-filling curve [144] of length* ϵ *that covers* $\mathbb{Z}^b_{0\,\cdots\,\lceil \frac{k}{b} \rceil}$*.* □

.

According to Definition 10, h is constructed as follows:

1. Using a hash function f (which can be treated as a black box for the moment), a record utilization vector $\vec{r}$ with k bits is divided into b disjoint segments $\vec{r}_0, \ldots, \vec{r}_{b-1}$ such that $\vec{r}_0, \ldots, \vec{r}_{b-1}$ contain all the bits in $\vec{r}$, and each $\vec{r}_0$, ..., $\vec{r}_{b-1}$ has at most $\lceil \frac{k}{b} \rceil$ bits. Then, a record utilization counter $\vec{c}$ with b entries is computed such that the i^{th} entry of $\vec{c}$ (i.e., $c[i]$) contains the number of 1-bits in $\vec{r}_i \in \{\vec{r}_0, \ldots, \vec{r}_{b-1}\}$. Without loss of generality, a record utilization counter $\vec{c}$ can be represented as a b-dimensional point in the coordinate system $\mathbb{Z}^b_{0 \cdots \lceil \frac{k}{b} \rceil}$.

2. The final hash value is determined by computing the z-value [80, 144] of the points in $\mathbb{Z}^b_{0 \cdots \lceil \frac{k}{b} \rceil}$, and dropping off the last m bits from the produced z-values, where $m = k - \lceil \log_2 \epsilon \rceil$.

In Section 4.3.3, it is shown that Tunable-LSH that maps k-dimensional record utilization vectors to natural numbers in the interval $[0, \ldots, \epsilon - 1]$ is locality-sensitive, with two important implications: (i) records with similar record utilization vectors (i.e., small Hamming distances) are likely going to be hashed to the same value, while (ii) records with dissimilar record utilization vectors are likely going to be separated. Therefore, the problem of computing G-by-Q clusters can be approximated using Tunable-LSH, such that clustering n records takes $\mathrm{O}(n)$ time.

The quality of Tunable-LSH, that is, how well it approximates the original Hamming distances, depends on two factors: (i) the characteristics of the workload so far, which is reflected by the bit distribution in the record utilization vectors, and (ii) the choice of f. In Section 4.3.4, it is demonstrated that f can be tuned to adapt to the changing patterns in record utilization vectors to maintain the approximation quality of Tunable-LSH at a steady and high level.

Algorithms 8–10 outline the computation of the outcome of Tunable-LSH outcome and methods for incrementally tuning the LSH function every time a query is executed. Note that there are two design considerations: (i) tuning should take constant-time, otherwise, there is no point in using a function, (ii) the memory overhead should be low because it would be desirable to maximize the allocation of memory to core database functionality. Consequently, instead of relying on record utilization vectors, the algorithm computes and incrementally maintains record utilization counters (cf., Algorithm 8) that are much easier to maintain and that have a much smaller memory footprint due to the fact that $b \ll k$. Then, whenever there is a need to compute the outcome of the LSH

Algorithm 8 Initialize

Ensure:
Record utilization counters are allocated and initialized

1: **procedure** INITIALIZE()
2: construct int $C[\omega][2b]$ ▷ For simplicity, C is allocated statically; however, in practice, it can be allocated dynamically to reduce memory footprint.
3: **for all** $i \in (0, \ldots, \omega - 1)$ **do**
4: **for all** $j \in (0, \ldots, 2b - 1)$ **do**
5: $C[i][j] \leftarrow 0$
6: **end for**
7: **end for**
8: **end procedure**

function for a given record, the HASH procedure is called with the *id* of the record, which in turn relies on h_2 to compute the hash value (cf., Algorithm 10).

The TUNE procedure (Algorithm 9) looks at the next query access vector, and updates f (line 2), which will be discussed in more detail in Section 4.3.4. Then it computes positions of records that have been accessed by that query (line 3), and increments the corresponding entries in the utilization counters of those records that have been accessed (line 8). To determine which entry to increment, the algorithm relies on h_1, hence, $f(t)$ (cf., Def. 10) and a shifting scheme. In line 11, old entries in record utilization counters are reset based on an approach that is discussed in Section 4.3.5. In that section, also the shifting scheme is discussed.

4.3.3 Properties of Tunable-LSH

This section discusses the locality-sensitive properties of $h = h_2 \circ h_1$ and demonstrates that h can be used for clustering the records. First, the relationship between record utilization vectors and the record utilization counters that are obtained by applying h_1 is shown.

Theorem 1 (Distance Bounds)**.** *Given a pair of record utilization vectors $\vec{r}_1$ and $\vec{r}_2$ with size k, let $\vec{c}_1$ and $\vec{c}_2$ denote two record utilization counters with size b such that $\vec{c}_1 = h_1(\vec{r}_1)$ and $\vec{c}_2 = h_1(\vec{r}_2)$ (cf., Definition 10). Furthermore, let $c_1[i]$ and $c_2[i]$ denote the i^{th} entry in*

Algorithm 9 Tune

Require:

$\vec{\mathbf{q_t}}$: query access vector produced at time t

Ensure:

Underlying data structures are updated and f is tuned such that the LSH function maintains a steady approximation quality

```
1: procedure TUNE(q_t)
2:     RECONFIGURE-F(q_t)
3:     for all i ∈ POSITIONAL(q_t) do
4:         loc ← f(t)
5:         if loc < (shift % b) then
6:             loc += b
7:         end if
8:         C[i][loc]++                          ▷ Increment record utilization counters based on the new query access pattern
9:         if t%⌈k/b⌉ = 0 then                  ▷ Reset "old" counters
10:            shift++
11:            C[i][(shift+b)%2b] ← 0
12:        end if
13:    end for
14: end procedure
```

$\vec{c}_1$ and $\vec{c}_2$, respectively. Then,

$$\delta(\vec{r_1}, \vec{r_2}) \geq \sum_{i=0}^{b-1} \left| c_1[i] - c_2[i] \right| \tag{4.3}$$

where $\delta(\vec{r_1}, \vec{r_2})$ represents the Hamming distance between $\vec{r}_1$ and $\vec{r}_2$.

Proof 1. *It is possible to prove Theorem 1 by induction on b.*

***Base case**: Theorem 1 holds when $b = 1$. According to Definition 10, when $b = 1$, $c_1[0]$ and $c_2[0]$ correspond to the total number of 1-bits in $\vec{r}_1$ and $\vec{r}_2$, respectively. Note that the Hamming distance between $\vec{r}_1$ and $\vec{r}_2$ will be smallest if and only if these two record utilization vectors are aligned on as many 1-bits as possible. In that case, they will differ in only $\left| c_1[0] - c_2[0] \right|$ bits, which corresponds to their Hamming distance. Consequently, Equation 4.3 holds for $b = 1$.*

Algorithm 10 Hash

Require:
id: id of record whose hash is being computed
Ensure:
Hash value is returned
1: **procedure** HASH(id)
2: **return** Z-VALUE(C[id]) ▷ Apply h_2
3: **end procedure**

Inductive step: *It needs to be shown that if Equation 4.3 holds for $b \leq \alpha$, where α is a natural number greater than or equal to 1, then it must also hold for $b = \alpha + 1$.*

Let $\Pi_f(\vec{r}, g)$ denote a record utilization vector $r' = (r'[0], \ldots, r'[k-1])$ such that for all $i \in \{0, \ldots, k-1\}$, $r'[i] = r[i]$ holds if $f(i) = g$, and $r'[i] = 0$ otherwise. Then,

$$\delta(\vec{r}_1, \vec{r}_2) = \sum_{g=0}^{b-1} \delta(\Pi_f(\vec{r}_1, g), \Pi_f(\vec{r}_2, g)). \tag{4.4}$$

That is, the Hamming distance between any two record utilization vectors is the summation of their individual Hamming distances within each group of bits that share the same hash value with respect to f. This property holds because f is a (total) function, and Π_f masks all the irrelevant bits. As an abbreviation, let $\delta_g = \delta(\Pi_f(\vec{r}_1, g), \Pi_f(\vec{r}_2, g))$. Then, due to the same reasoning as in the base case, for $g = \alpha$, the following equation holds:

$$\delta_\alpha(\vec{r_1}, \vec{r_2}) \geq \left| c_1[\alpha] - c_2[\alpha] \right| \tag{4.5}$$

Consequently, using the additive property in Equation 4.4, it can be shown that Equation 4.3 holds also for $b = \alpha + 1$. Thus, by induction, Theorem 1 holds. □

Theorem 1 suggests that the Hamming distance between any two record utilization vectors $\vec{r_1}$ and $\vec{r_2}$ can be approximated using record utilization counters $\vec{c_1} = h_1(\vec{r_1})$ and $\vec{c_2} = h_1(\vec{r_2})$ because Equation 4.3 provides a lower bound on $\delta(\vec{r_1}, \vec{r_2})$. In fact, the right-hand side of Equation 4.3 is equal to the Manhattan distance [128] between $\vec{c_1}$ and $\vec{c_2}$ in $\mathbb{Z}^b_{0 \cdots \lceil \frac{k}{b} \rceil}$, and since $\delta(\vec{r_1}, \vec{r_2})$ is equal to the Manhattan distance between $\vec{r_1}$ and $\vec{r_2}$ in $\mathbb{Z}^k_{0 \cdots 1}$, it is easy to see that h_1 is a transformation that approximates Manhattan distances. The following corollary captures this property.

Corollary 2 (Distance Approximation)**.** *Given a pair of record utilization vectors $\vec{r}_1$ and*

$\vec{r}_2$ with size k, let $\vec{c}_1$ and $\vec{c}_2$ denote two points in the coordinate system $\mathbb{Z}^b_{0\cdots\lceil\frac{k}{b}\rceil}$ such that $\vec{c}_1 = h_1(\vec{r}_1)$ and $\vec{c}_2 = h_1(\vec{r}_2)$ (cf., Definition 10). Let $\delta^M(\vec{r}_1, \vec{r}_2)$ denote the Manhattan distance between $\vec{r}_1$ and $\vec{r}_2$, and let $\delta^M(\vec{c}_1, \vec{c}_2)$ denote the Manhattan distance between $\vec{c}_1$ and $\vec{c}_2$. Then, the following holds:

$$\delta(\vec{r}_1, \vec{r}_2) = \delta^M(\vec{r}_1, \vec{r}_2) \geq \delta^M(\vec{c}_1, \vec{c}_2) \tag{4.6}$$

Proof 2. *Hamming distance in $\mathbb{Z}^k_{0\cdots 1}$ is a special case of Manhattan distance. Furthermore, by definition [128], the right hand side of Equation 4.3 equals the Manhattan distance $\delta^M(\vec{c}_1, \vec{c}_2)$; therefore, Equation 4.6 holds.* □

Next, it is demonstrated that $h = h_2 \circ h_1$ is a locality-sensitive transformation [83,116]. In particular, the definition of locality-sensitiveness by Tao et al. [180] is used, and it is shown that the probability that two record utilization vectors $\vec{r}_1$ and $\vec{r}_2$ are hashed to the same page increases as the (Manhattan) distance between r_1 and r_2 decreases.

Theorem 3 (Collision Probabilities)**.** *Given a pair of record utilization vectors $\vec{r}_1$ and $\vec{r}_2$ with size k, let $\delta^M(\vec{r}_1, \vec{r}_2)$ denote the Manhattan distance between $\vec{r}_1$ and $\vec{r}_2$. Furthermore, let m denote the number of rightmost bits that are dropped by h_2. When $b = 1$ (i.e., the size of the record utilization counters produced by h_1), the probability that* the pair of record utilization vectors will be hashed to the same value by $h_2 \circ h_1$ provided that their initial Manhattan distance is x *is given by the following formula:*

$$\begin{aligned} &\Pr\Big(h_2 \circ h_1(\vec{r}_1) = h_2 \circ h_1(\vec{r}_2) \;\Big|\; \delta^M(\vec{r}_1, \vec{r}_2) = x\Big) \\ &\quad = \frac{\sum_{a=0}^{x} \sum_{\Delta=0}^{k-x} \binom{x}{a}\binom{k}{k-x}\binom{k-x}{\Delta}\rho(\Delta + a, x - 2a, m)}{2^{2k}} \end{aligned} \tag{4.7}$$

where $\rho(x, y, m) : (\mathbb{Z}_{0\cdots\infty}, \mathbb{Z}_{-\infty\cdots\infty}, \mathbb{Z}_{0\cdots\infty}) \to \{0, 1\}$ is a function such that

$$\rho(x, y, m) = \begin{cases} 1 & \text{if } 0 \leq (x \mod 2^m) + y < 2^m \\ 0 & \text{else} \end{cases}.$$

Proof 3. *If the Hamming/Manhattan distance between $\vec{r}_1$ and $\vec{r}_2$ is x, then it means that*

these two vectors will differ in exactly x bits, as shown below.

$$\vec{\mathbf{r_1}}: \square\square\square \overbrace{\mathbf{111\ldots1}}^{a} \mathbf{0\cdots000}\square\square\square$$
$$\vec{\mathbf{r_2}}: \square\square\square \mathbf{000\cdots0} \underbrace{\mathbf{1\ldots111}}_{x-a} \square\square\square$$

Furthermore, if $\vec{r_1}$ has $\Delta + a$ bits set to 1, then $\vec{r_2}$ must have $\Delta + (x - a)$ bits set to 1, where Δ denotes the number of matching 1-bits between $\vec{r_1}$ and $\vec{r_2}$. Note that when $b = 1$, $\vec{c_1} = h_1(\vec{r_1}) = (\Delta + a)$ and $\vec{c_2} = h_1(\vec{r_2}) = (\Delta + x - a)$.

It is easy to see that $a \in \mathbb{Z}_{0\ldots x}$ and $\Delta \in \mathbb{Z}_{0\ldots k-x}$. For each value of a, the non-matching bits in $\vec{r_1}$ and $\vec{r_2}$ can be combined in $\binom{x}{a}$ possible ways, and these non-matching bits can be positioned across the k bits in $\binom{k}{k-x}$ possible ways. Likewise, for each value of Δ, those matching 1-bits that are counted by Δ can be combined in $\binom{k-x}{\Delta}$ possible ways, hence, the first three components of the multiplication in the numerator of Equation 4.7.

Among the aforementioned combinations, $h_2(\vec{c_1}) = h_2(\vec{c_2})$ will be true if and only if the binary representations of $\vec{c_1}$ and $\vec{c_2}$ share the same sequence of bits except for their last m bits. This condition will be satisfied if and only if $\Delta + a$ and $\Delta + x - a$ have the same quotient when divided by 2^m. In other words, $(\Delta + a) \mod 2^m + (\Delta + x - a) - (\Delta + a)$ must be greater than or equal to 0 or less than 2^m, hence, the need to multiply by ρ in the numerator of Equation 4.7.

Since $\vec{r_1}$ and $\vec{r_2}$ consist of k bits, there can be $2^k \times 2^k = 2^{2k}$ possible combinations in total, which corresponds to the denominator of Equation 4.7. □

Next, Theorem 3 is extended to cases in which $b \geq 2$. However, first, some auxiliary statements need to be made.

Lemma 4. *Let $\rho(x, y, m) : (\mathbb{Z}_{0\ldots\infty}, \mathbb{Z}_{-\infty\ldots\infty}, \mathbb{Z}_{0\ldots\infty}) \rightarrow \{0, 1\}$ denote a function such that*

$$\rho(x, y, m) = \begin{cases} 1 & \text{if } 0 \leq (x \mod 2^m) + y < 2^m \\ 0 & \text{else} \end{cases}.$$

Then, for any $m \in \mathbb{Z}_{0\ldots\infty}$, the following properties hold:

Property A. For any $x \in \mathbb{Z}_{0\ldots\infty}$, if $0 \leq y_1 \leq y_2$, then $\rho(x, y_1, m) \geq \rho(x, y_2, m)$;

Property B. For any $x \in \mathbb{Z}_{0\ldots\infty}$, if $y_2 \leq y_1 \leq 0$, then $\rho(x, y_1, m) \geq \rho(x, y_2, m)$;

Property C. *For any x_1, x_2, y_1, y_2 such that $x_1, x_2 \in \mathbb{Z}_{0\ldots\infty}$ and $x_1 + y_1 = x_2 + y_2$, if $0 \leq y_1 \leq y_2$, then $\rho(x_1, y_1, m) \geq \rho(x_2, y_2, m)$; and*

Property D. *For any x_1, x_2, y_1, y_2 such that $x_1, x_2 \in \mathbb{Z}_{0\ldots\infty}$ and $x_1 + y_1 = x_2 + y_2$, if $y_2 \leq y_1 \leq 0$, then $\rho(x_1, y_1, m) \geq \rho(x_2, y_2, m)$; and*

Proof 4. *All of the four properties are proven by contradiction.*

Property A can be proven as follows:

A1. *For any $x \in \mathbb{Z}_{0\ldots\infty}$ and $m \in \mathbb{Z}_{0\ldots\infty}$,*

$$0 \leq x \mod 2^m < 2^m \tag{4.8}$$

by the definition of the modulo operation.

A2. *Assume $0 \leq y_1 \leq y_2$.*

- *Assume $\rho(x, y_1, m) < \rho(x, y_2, m)$.*
- *Therefore, since $0 \leq y_1 \leq y_2$,*

$$0 \leq x \mod 2^m + y_1 \leq x \mod 2^m + y_2. \tag{4.9}$$

- *Since $\rho(x, y_1, m) < \rho(x, y_2, m)$, $\rho(x, y_1, m) = 0$ and $\rho(x, y_2, m) = 1$. (Note that the codomain of ρ is the set $\{0, 1\}$.)*
- *If $\rho(x, y_1, m) = 0$, according to the definition of ρ (cf., Lemma 4) and Equation 4.8, and based on the fact that $y_1 \geq 0$, the following statement must be true:*

$$x \mod 2^m + y_1 \geq 2^m. \tag{4.10}$$

- *Likewise, if $\rho(x, y_2, m) = 1$, according to the definition of ρ (cf., Lemma 4), the following statement must be true:*

$$0 \leq x \mod 2^m + y_2 < 2^m. \tag{4.11}$$

- *Therefore, according to Equations 4.10 and 4.11,*

$$x \mod 2^m + y_2 < x \mod 2^m + y_1. \tag{4.12}$$

- *However, Equation 4.12 contradicts Equation 4.9, therefore,*

$$\rho(x, y_1, m) \geq \rho(x, y_2, m). \tag{4.13}$$

A3. Since Equation 4.13 holds under the assumption that $0 \leq y_1 \leq y_2$*, Property A in Lemma 4 holds.*

Property B can be proven as follows:

B1. For any $x \in \mathbb{Z}_{0\ldots\infty}$ *and* $m \in \mathbb{Z}_{0\ldots\infty}$*,*

$$0 \leq x \mod 2^m < 2^m \tag{4.14}$$

by the definition of the modulo operation.

B2. Assume $y_2 \leq y_1 \leq 0$*.*

- *Assume* $\rho(x, y_1, m) < \rho(x, y_2, m)$*.*
- *Therefore, since* $y_2 \leq y_1 \leq 0$*,*

$$x \mod 2^m + y_2 \leq x \mod 2^m + y_1 < 2^m. \tag{4.15}$$

- *Since* $\rho(x, y_1, m) < \rho(x, y_2, m)$*,* $\rho(x, y_1, m) = 0$ *and* $\rho(x, y_2, m) = 1$*. (Note that the codomain of* ρ *is the set* $\{0, 1\}$*.)*
- *If* $\rho(x, y_1, m) = 0$*, according to the definition of* ρ *(cf., Lemma 4) and Equation 4.14, and based on the fact that* $y_1 \leq 0$*, the following statement must be true:*

$$x \mod 2^m + y_1 < 0. \tag{4.16}$$

- *Likewise, if* $\rho(x, y_2, m) = 1$*, according to the definition of* ρ *(cf., Lemma 4), the following statement must be true:*

$$0 \leq x \mod 2^m + y_2 < 2^m. \tag{4.17}$$

- *Therefore, according to Equations 4.16 and 4.17,*

$$x \mod 2^m + y_1 < x \mod 2^m + y_2. \tag{4.18}$$

- *However, Equation 4.18 contradicts Equation 4.15, therefore,*

$$\rho(x, y_1, m) \geq \rho(x, y_2, m). \tag{4.19}$$

B3. Since Equation 4.19 holds under the assumption that $y_2 \leq y_1 \leq 0$, Property B in Lemma 4 holds.

Property C can be proven as follows:

C1. For any $x_1, x_2 \in \mathbb{Z}_{0\ldots\infty}$ and $m \in \mathbb{Z}_{0\ldots\infty}$,

$$\begin{aligned} 0 \leq x_1 \ \ mod\ 2^m < 2^m \\ 0 \leq x_2 \ \ mod\ 2^m < 2^m \end{aligned} \tag{4.20}$$

by the definition of the modulo operation.

C2. Assume

$$\begin{aligned} 0 \leq y_1 \leq y_2 \ and \\ x_1 + y_1 = x_2 + y_2. \end{aligned} \tag{4.21}$$

- *Assume $\rho(x_1, y_1, m) < \rho(x_2, y_2, m)$.*
- *Since $\rho(x_1, y_1, m) < \rho(x_2, y_2, m)$, $\rho(x_1, y_1, m) = 0$ and $\rho(x_2, y_2, m) = 1$. (Note that the codomain of ρ is the set $\{0, 1\}$.)*
- *If $\rho(x_1, y_1, m) = 0$, according to the definition of ρ (cf., Lemma 4) and Equation 4.20, and based on the fact that $y_1 \geq 0$, the following statement must be true:*

$$x_1 \ \ mod\ 2^m + y_1 \geq 2^m. \tag{4.22}$$

- *Likewise, if $\rho(x_2, y_2, m) = 1$, according to the definition of ρ (cf., Lemma 4), the following statement must be true:*

$$0 \leq x_2 \ \ mod\ 2^m + y_2 < 2^m. \tag{4.23}$$

- *Since $x_1 = x_2 + y_2 - y_1$ (cf., Equation 4.21), the following statements are true:*

$$\begin{aligned} x_1 \ \ mod\ 2^m &= (x_2 + y_2 - y_1) \ \ mod\ 2^m \\ &= (x_2 \ \ mod\ 2^m + (y_2 - y_1) \ \ mod\ 2^m) \ \ mod\ 2^m. \end{aligned} \tag{4.24}$$

- *Note that*

$$x_2 \ mod\ 2^m + (y_2 - y_1) \ mod\ 2^m \ \geq \ (x_2 \ mod\ 2^m + (y_2 - y_1) \ mod\ 2^m) \ mod\ 2^m$$

(i.e., the mod 2^m of any positive number is always less than or equal to the number itself).

- *Therefore, Equation 4.24 can be restated as*

$$x_1 \ mod\ 2^m \ \leq \ x_2 \ mod\ 2^m + (y_2 - y_1) \ mod\ 2^m.$$

- *Likewise, the following statement must be true:*

$$x_1 \ mod\ 2^m \ \leq \ x_2 \ mod\ 2^m + (y_2 - y_1) \tag{4.25}$$

because $(y_2 - y_1) \geq 0$, therefore, $y_2 - y_1$ is always greater than or equal to its modulo in 2^m.

- *Therefore,*

$$x_1 \ mod\ 2^m + y_1 \ \leq \ x_2 \ mod\ 2^m + y_2.$$

- *It is also known from Equation 4.23 that $x_2 mod 2^m < 2^m$, therefore $x_1 mod 2^m < 2^m$ must also be true. This, however, contradicts Equation 4.22, therefore,*

$$\rho(x_1, y_1, m) \geq \rho(x_2, y_2, m). \tag{4.26}$$

C3. Since Equation 4.26 holds under the assumption that $0 \leq y_1 \leq y_2$ and $x_1 + x_2 = y_1 + y_2$, Property C in Lemma 4 holds.

Property D can be proven as follows:

D1. For any $x_1, x_2 \in \mathbb{Z}_{0 \ldots \infty}$ and $m \in \mathbb{Z}_{0 \ldots \infty}$,

$$\begin{aligned} 0 \leq x_1 \ mod\ 2^m < 2^m \\ 0 \leq x_2 \ mod\ 2^m < 2^m \end{aligned} \tag{4.27}$$

by the definition of the modulo operation.

D2. Assume

$$y_2 \leq y_1 \leq 0 \text{ and}$$
$$x_1 + y_1 = x_2 + y_2. \tag{4.28}$$

- *Assume* $\rho(x_1, y_1, m) < \rho(x_2, y_2, m)$.
- *Since* $\rho(x_1, y_1, m) < \rho(x_2, y_2, m)$, $\rho(x_1, y_1, m) = 0$ *and* $\rho(x_2, y_2, m) = 1$. *(Note that the codomain of* ρ *is the set* $\{0, 1\}$*.)*
- *If* $\rho(x_1, y_1, m) = 0$, *according to the definition of* ρ *(cf., Lemma 4) and Equation 4.27, and based on the fact that* $y_1 \leq 0$, *the following statement must be true:*

$$x_1 \ mod\ 2^m + y_1 < 0. \tag{4.29}$$

- *Likewise, if* $\rho(x_2, y_2, m) = 1$, *according to the definition of* ρ *(cf., Lemma 4), the following statement must be true:*

$$0 \leq x_2 \ mod\ 2^m + y_2 < 2^m. \tag{4.30}$$

- *Since* $x_2 = x_1 + y_1 - y_2$ *(cf., Equation 4.28), the following statements are true:*

$$\begin{aligned} x_2 \ mod\ 2^m &= (x_1 + y_1 - y_2) \ mod\ 2^m \\ &= (x_1 \ mod\ 2^m + (y_1 - y_2) \ mod\ 2^m) \ mod\ 2^m. \end{aligned} \tag{4.31}$$

- *Then, for the same reasons discussed in the proof of Property C,*

$$\begin{aligned} x_2 \ mod\ 2^m &\leq x_1 \ mod\ 2^m + (y_1 - y_2) \ mod\ 2^m \\ x_2 \ mod\ 2^m &\leq x_1 \ mod\ 2^m + y_1 - y_2 \\ x_2 \ mod\ 2^m + y_2 &\leq x_1 \ mod\ 2^m + y_1. \end{aligned} \tag{4.32}$$

- *Since* $x_1 \ mod\ 2^m + y_1 < 0$ *(cf., Equation 4.29), according to Equation 4.32,* $x_2 \bmod 2^m + y_2 < 0$ *must also hold, but this statement contradicts Equation 4.30. Therefore,*

$$\rho(x_1, y_1, m) \geq \rho(x_2, y_2, m). \tag{4.33}$$

D3. Since Equation 4.33 holds under the assumption that $y_2 \leq y_1 \leq 0$ *and* $x_1 + x_2 = y_1 + y_2$, *Property D in Lemma 4 holds.* □

Lemma 5. *Let $\delta^M(\vec{r}_i, \vec{r}_j)$ denote the Manhattan distance between any two record utilization vectors $\vec{r}_i$ and $\vec{r}_j$, and let m denote the number of rightmost bits that are dropped by h_2, where h_2 is utilized in h (cf., Definition 10). Furthermore, let $\text{PR}_{==}\big(\vec{r}_i, \vec{r}_j, x\big)$ denote the posterior probability that, for any two record utilization vectors $\vec{r}_i$ and $\vec{r}_j$, $h(\vec{r}_i) = h(\vec{r}_j)$ provided that $\delta^M(\vec{r}_i, \vec{r}_j) = x$. Given any four record utilization vectors $\vec{r}_1$, $\vec{r}_2$, $\vec{r}_3$ and $\vec{r}_4$ with size k such that k is even, $\delta^M(\vec{r}_1, \vec{r}_2) = x$ and $\delta^M(\vec{r}_3, \vec{r}_4) = k - x$, the following property holds for any $x \leq \lfloor \frac{k}{2} \rfloor$ and $m \in \mathbb{Z}_{0 \ldots k-1}$:*

$$\text{PR}_{==}\big(\vec{r}_1, \vec{r}_2, x\big) \geq \text{PR}_{==}\big(\vec{r}_3, \vec{r}_4, k - x\big) \tag{4.34}$$

where $h = h_2 \circ h_1$ (cf., Definition 10), and $b = 1$ denotes the number of entries in the record utilization counters produced by h_1.

Proof 5. *The proof of Lemma 5 proceeds in multiple steps.*

S1. It is shown that for any two natural numbers a and Δ such that $a \leq \frac{x}{2}$, $\Delta \leq (k - x)$ and $\Delta - a \leq \frac{k-2x}{2}$, the following property holds:

$$\begin{aligned} &\binom{x}{a}\binom{k-x}{\Delta}\rho(a + \Delta, x - 2a) \geq \\ &\qquad \binom{k-x}{\Delta}\binom{x}{a}\rho(a + \Delta, k - x - 2\Delta). \end{aligned} \tag{4.35}$$

This statement can be proven as follows:

- *For any two natural numbers a and Δ such that $a \leq \frac{x}{2}$ and $\Delta \leq (k - x)$, the binomials are defined and their products are equal (and positive) on both sides of the inequality.*
- *The condition $a \leq \frac{x}{2}$ in S1 implies that $0 \leq (x - 2a)$.*
- *The condition $\Delta - a \leq \frac{k-2x}{2}$ in S1 implies that*

$$\begin{aligned} \Delta - a &\leq \frac{k - 2x}{2} \\ 2(\Delta - a) &\leq k - 2x \\ 2\Delta - 2a &\leq k - 2x \\ x - 2a &\leq k - x - 2\Delta. \end{aligned}$$

- *According to Property A in Lemma 4, $\rho(a + \Delta, x - 2a) \geq \rho(a + \Delta, k - x - 2\Delta)$. Therefore, statement S1 holds.*

S2. It is shown that for any two natural numbers a and Δ such that $\frac{x}{2} \leq a \leq x$, $\Delta \leq (k-x)$ and $\Delta - a \geq \frac{k-2x}{2}$, the following property holds:

$$\binom{x}{a}\binom{k-x}{\Delta}\rho(a+\Delta,\, x-2a) \geq \binom{k-x}{\Delta}\binom{x}{a}\rho(a+\Delta,\, k-x-2\Delta). \tag{4.36}$$

This statement can be proven as follows:

- *For any two natural numbers a and Δ such that $\frac{x}{2} \leq a \leq x$ and $\Delta \leq (k-x)$, the binomials are defined and their products are equal (and positive) on both sides of the inequality.*
- *The condition $a \geq \frac{x}{2}$ in S2 implies that $(x-2a) \leq 0$.*
- *The condition $\Delta - a \geq \frac{k-2x}{2}$ in S2 implies that*

$$\begin{aligned} \Delta - a &\geq \frac{k-2x}{2} \\ 2(\Delta - a) &\geq k-2x \\ 2\Delta - 2a &\geq k-2x \\ x-2a &\geq k-x-2\Delta. \end{aligned}$$

- *According to Property B in Lemma 4, $\rho(a+\Delta,\, x-2a) \geq \rho(a+\Delta,\, k-x-2\Delta)$. Therefore, statement S2 holds.*

S3. It is shown that for any two natural numbers a and Δ such that $a \leq \frac{x}{2}$, $\Delta \leq (k-x)$, and $\Delta + a \geq \frac{k}{2}$, the following property holds:

$$\binom{x}{a}\binom{k-x}{\Delta}\rho(a+\Delta,\, x-2a) \geq \binom{k-x}{k-x-\Delta}\binom{x}{x-a}\rho(k-\Delta-a,\, x-k+2\Delta). \tag{4.37}$$

This statement can be proven as follows:

- *For any two natural numbers a and Δ such that $a \leq \frac{x}{2}$ and $\Delta \leq (k-x)$, the binomials are defined and their products are equal (and positive) on both sides of the inequality because*

  - $\binom{x}{a} = \binom{x}{x-a}$, *and*
  - $\binom{k-x}{\Delta} = \binom{k-x}{k-x-\Delta}$.
- *Note that* $(a+\Delta)+(x-2a) = (k-\Delta-a)+(x-k+2\Delta)$ *is true as shown below:*

$$\begin{aligned}(a+\Delta)+(x-2a) &\stackrel{?}{=} (k-\Delta-a)+(x-k+2\Delta)\\ x+\Delta-a &= x+\Delta-a.\end{aligned}$$

- *The condition* $a \leq \frac{x}{2}$ *in S3 implies that* $0 \leq (x-2a)$.
- *The condition* $\Delta + a \geq \frac{k}{2}$ *in S3 implies that* $(x-2a) \leq (x-k+2\Delta)$ *as shown below:*

$$\begin{aligned}\Delta + a &\geq \frac{k}{2}\\ -2(\Delta+a) &\leq -k\\ -2\Delta - 2a &\leq -k\\ -2a &\leq -k+2\Delta\\ x-2a &\leq x-k+2\Delta.\end{aligned}$$

- *According to Property C in Lemma 4,* $\rho(a+\Delta,\, x-2a) \geq \rho(k-\Delta-a,\, x-k+2\Delta)$ *is true because* $(a+\Delta)+(x-2a) = (k-\Delta-a)+(x-k+2\Delta)$ *and* $0 \leq (x-2a) \leq (x-k+2\Delta)$. *Therefore, statement S3 must be true.*

S4. It is shown that for any two natural numbers a *and* Δ *such that* $\frac{x}{2} \leq a \leq x$, $\Delta \leq (k-x)$, *and* $\Delta + a \leq \frac{k}{2}$, *the following property holds:*

$$\begin{aligned}&\binom{x}{a}\binom{k-x}{\Delta}\rho(a+\Delta,\, x-2a) \geq\\ &\qquad\binom{k-x}{k-x-\Delta}\binom{x}{x-a}\rho(k-\Delta-a, x-k+2\Delta).\end{aligned} \tag{4.38}$$

This statement can be proven as follows:

- *For any two natural numbers* a *and* Δ *such that* $\frac{x}{2} \leq a \leq x$ *and* $\Delta \leq (k-x)$, *the binomials are defined and their products are equal (and positive) on both sides of the inequality because*
  - $\binom{x}{a} = \binom{x}{x-a}$, *and*

$-\binom{k-x}{\Delta} = \binom{k-x}{k-x-\Delta}$.

- *Note that* $(a+\Delta)+(x-2a) = (k-\Delta-a)+(x-k+2\Delta)$ *is true as shown below:*

$$\begin{aligned}(a+\Delta)+(x-2a) &\stackrel{?}{=} (k-\Delta-a)+(x-k+2\Delta)\\ x+\Delta-a &= x+\Delta-a.\end{aligned}$$

- *The condition* $a \geq \frac{x}{2}$ *in S4 implies that* $(x-2a) \leq 0$.
- *The condition* $\Delta + a \leq \frac{k}{2}$ *in S4 implies that* $(x-2a) \geq (x-k+2\Delta)$ *as shown below:*

$$\begin{aligned}\Delta + a &\leq \frac{k}{2}\\ -2(\Delta+a) &\geq -k\\ -2\Delta-2a &\geq -k\\ -2a &\geq -k+2\Delta\\ x-2a &\geq x-k+2\Delta.\end{aligned}$$

- *According to Property D in Lemma 4,* $\rho(a+\Delta,\, x-2a) \geq \rho(k-\Delta-a,\, x-k+2\Delta)$ *is true because* $(a+\Delta)+(x-2a) = (k-\Delta-a)+(x-k+2\Delta)$ *and* $(x-k+2\Delta) \leq (x-2a) \leq 0$. *Therefore, statement S4 must be true.*

S5. It is shown that for any two natural numbers a *and* Δ *such that* $a \leq \frac{x}{2}$, $\Delta \leq (k-x)$, $\Delta - a > \frac{k-2x}{2}$, *and* $\Delta + a < \frac{k}{2}$, *if* $\rho(a+\Delta,\, x-2a) < \rho(a+\Delta,\, k-x-2\Delta)$, *there exists two natural numbers* a' *and* Δ' *such that*

$$0 \leq a' = \Delta - \frac{k-2x}{2} \leq x \qquad \textit{and} \qquad 0 \leq \Delta' = a + \frac{k-2x}{2} \leq k-x,$$

and the following property holds:

$$\begin{aligned}&\binom{x}{a}\binom{k-x}{\Delta}\rho(a+\Delta,\, x-2a) + \binom{x}{a'}\binom{k-x}{\Delta'}\rho(a'+\Delta',\, x-2a') \geq\\ &\binom{k-x}{\Delta}\binom{x}{a}\rho(a+\Delta,\, k-x-2\Delta) + \binom{k-x}{\Delta'}\binom{x}{a'}\rho(a'+\Delta',\, k-x-2\Delta'). \quad (4.39)\end{aligned}$$

The statement is proven by showing that the following statements hold under the conditions given in S5:

- $0 \leq a' \leq x$,

- $0 \leq \Delta' \leq k - x$,
- $\rho(a + \Delta,\, x - 2a) = \rho(a' + \Delta',\, k - x - 2\Delta')$,
- $\rho(a' + \Delta',\, x - 2a') = \rho(a + \Delta,\, k - x - 2\Delta)$, *and*
- $\binom{x}{a}\binom{k-x}{\Delta} \leq \binom{k-x}{\Delta'}\binom{x}{a'}$.

The proof steps are as follows:

- $a' \geq 0$ *holds because*

$$\begin{aligned}
\Delta - a &> \frac{k - 2x}{2} && \textit{cf., S5} \\
\Delta - \frac{k - 2x}{2} &> a \\
a' &> a && \textit{cf., S5} \\
a' &\geq 0 && \textit{since } a \geq 0.
\end{aligned}$$

- $a' \leq x$ *holds because*

$$\begin{aligned}
\Delta + a &< \frac{k}{2} && \textit{cf., S5} \\
\Delta &< \frac{k}{2} - a \\
\Delta - \left(\frac{k - 2x}{2}\right) &< \frac{k}{2} - a - \left(\frac{k - 2x}{2}\right) \\
\Delta - \left(\frac{k - 2x}{2}\right) &< \frac{\cancel{k}}{\cancel{2}} - a - \frac{\cancel{k}}{\cancel{2}} + x \\
\Delta - \left(\frac{k - 2x}{2}\right) &< x - a \\
a' &< x - a \\
a' &< x - a \leq x && \textit{since } a \geq 0 \\
a' &\leq x.
\end{aligned}$$

- $\Delta' \geq 0$ *holds because*

$$\begin{aligned} \frac{k}{2} &\geq x && \textit{cf., Lemma 5} \\ \frac{k}{2} - x &\geq 0 \\ \frac{k-2x}{2} &\geq 0 \\ a + \frac{k-2x}{2} &\geq 0 && \textit{since } a \geq 0 \\ \Delta' &\geq 0. \end{aligned}$$

- $\Delta' \leq k - x$ *holds because*

$$\begin{aligned} \Delta - a &> \frac{k-2x}{2} && \textit{cf., S5} \\ \Delta &> a + \frac{k-2x}{2} \\ k - x \geq \Delta &> a + \frac{k-2x}{2} && \textit{since } \Delta \leq k - x \\ k - x &\geq a + \frac{k-2x}{2} \\ k - x &\geq \Delta'. \end{aligned}$$

- $\rho(a + \Delta,\, x - 2a) = \rho(a' + \Delta',\, k - x - 2\Delta')$ *holds because* $a + \Delta = a' + \Delta'$ *and* $x - 2a = k - x - 2\Delta'$*, as shown below:*

$$\begin{aligned} a + \Delta &\stackrel{?}{=} a' + \Delta' \\ a + \Delta &\stackrel{?}{=} \Delta - \cancel{\left(\frac{k-2x}{2}\right)} + a + \cancel{\left(\frac{k-2x}{2}\right)} && \textit{cf., S5} \\ a + \Delta &= \Delta + a, \end{aligned}$$

and

$$\begin{aligned} x - 2a &\stackrel{?}{=} k - x - 2\Delta' \\ x - 2a &\stackrel{?}{=} \cancel{k} - x - 2a - \cancel{k} + 2x \qquad \textit{since } \Delta' = a + \frac{k-2x}{2} \\ x - 2a &= x - 2a. \end{aligned}$$

- $\rho(a' + \Delta', x - 2a') = \rho(a + \Delta, k - x - 2\Delta)$ *holds because* $a + \Delta = a' + \Delta'$*, as shown above, and* $x - 2a' = k - x - 2\Delta$*, as shown below:*

$$\begin{aligned} x - 2a' &\stackrel{?}{=} k - x - 2\Delta \\ x - 2\Delta + 2\Delta' - 2a &\stackrel{?}{=} k - x - 2\Delta \qquad \textit{since } a' = \Delta - \Delta' + a \\ x - 2\Delta + \cancel{2a} + k - 2x - \cancel{2a} &\stackrel{?}{=} k - x - 2\Delta \qquad \textit{since } \Delta' = a + \frac{k-2x}{2} \\ x - 2\Delta + k - 2x &\stackrel{?}{=} k - x - 2\Delta \\ k - x - 2\Delta &= k - x - 2\Delta. \end{aligned}$$

- *Let* $M = a' - a = \Delta - \Delta' > 0$ *and* $N = \Delta - a' = \Delta' - a = \frac{k-2x}{2} \geq 0$ *(conditions trivially follow from earlier proof steps). Then,* $\binom{x}{a}\binom{k-x}{\Delta} \leq \binom{k-x}{\Delta'}\binom{x}{a'}$ *is true, as shown below:*

$$\binom{x}{a}\binom{k-x}{\Delta} = \frac{x!}{a!(x-a)!} \times \frac{(k-x)!}{(a+M+N)!(k-x-a-M-N)!}$$

and

$$\binom{x}{a'}\binom{k-x}{\Delta'} = \frac{x!}{(a+M)!(x-a-M)!} \times \frac{(k-x)!}{(a+N)!(k-x-a-N)!}.$$

Since the numerators are the same (and positive) in both of the equations above, $\binom{x}{a}\binom{k-x}{\Delta} \leq \binom{k-x}{\Delta'}\binom{x}{a'}$ *iff*

$$\frac{a!(x-a)!}{(a+M)!(x-a-M)!} \times \frac{(a+M+N)!(k-x-a-M-N)!}{(a+N)!(k-x-a-N)!} \geq 1.$$

By expanding the factorials, one obtains

$$\frac{\overbrace{(x-a)\times\cdots\times(x-a-M+1)}^{M\ times}}{\underbrace{(a+M)\times\cdots\times(a+1)}_{M\ times}}\times\frac{\overbrace{(a+N+M)\times\cdots\times(a+N+1)}^{M\ times}}{\underbrace{(k-x-a-N)\times\cdots\times(k-x-a-N-M+1)}_{M\ times}}\geq 1.$$

The inequality above can be simplified further, as shown below:

$$\prod_{i=1}^{M}\frac{(a+N+i)}{(a+i)}\times\frac{(x+1)-(a+i)}{(k-x+1)-(a+N+i)}\geq 1. \tag{4.40}$$

Note that

$$\begin{aligned}(a+N+i)+(x+1)-(a+i)&=(a+i)+(k-x+1)-(a+N+i)\\ N+x+1&=k-x+1-N\\ \frac{k}{2}-x+x+1&=k-x+1-\frac{k}{2}+x\quad \textit{since } N=\frac{k-2x}{2}\\ \frac{k}{2}+1&=\frac{k}{2}+1.\end{aligned}$$

Therefore, Inequality 4.40 can be restated as:

$$\prod_{i=1}^{M}\frac{\big(a+N+i\big)}{\big(a+i\big)}\times\frac{\big((\frac{k}{2}+1)-(a+N+i)\big)}{\big((\frac{k}{2}+1)-(a+i)\big)}\geq 1. \tag{4.41}$$

Note that Inequality 4.41 holds iff

$$\begin{aligned}\Big(\frac{k}{2}+1-\big((a+1)+(a+M)\big)\Big)^2&\geq\Big(\frac{k}{2}+1-\big((a+N+1)+(a+N+M)\big)\Big)^2\\ \Big(\frac{k}{2}+1-\big(2a+M+1\big)\Big)^2&\geq\Big(\frac{k}{2}+1-\big(2a+M+1+2N\big)\Big)^2\\ \Big(\frac{k}{2}-2a-M\Big)^2&\geq\Big((\frac{k}{2}-2a-M)-2N\Big)^2.\end{aligned}$$

Let $\xi = \frac{k}{2} - 2a - M$*, then, the inequality can be restated as:*

$$\begin{aligned}
\Big(\xi\Big)^2 &\geq \Big(\xi - 2N\Big)^2 \\
\cancel{\xi^2} &\geq \cancel{\xi^2} - 4N\xi + 4N^2 \\
\cancel{4N}\xi &\geq \cancel{4N} \times N && \text{since } N \geq 0 \\
\frac{k}{2} - 2a - M &\geq N \\
\frac{\cancel{k}}{\cancel{2}} - 2a - M &\geq \frac{\cancel{k}}{\cancel{2}} - x && \text{since } N = \frac{k-2x}{2} \\
2a + M &\leq x. && (4.42)
\end{aligned}$$

Note that $M = \Delta - \Delta' = \Delta - a - \frac{k}{2} + x$*. Therefore,*

$$\begin{aligned}
2a + M &= 2a + \Delta - a - \frac{k}{2} + x \\
&= a + \Delta - \frac{k}{2} + x.
\end{aligned}$$

Substituting for $2a + M$ *in Inequality 4.42, it is possible to obtain*

$$\begin{aligned}
a + \Delta - \frac{k}{2} + \cancel{x} &\leq \cancel{x} \\
a + \Delta &\leq \frac{k}{2}. && (4.43)
\end{aligned}$$

Since $a + \Delta < \frac{k}{2}$ *is a precondition of S5, Inequalities 4.40–4.43 must be true. Consequently,* $\binom{x}{a}\binom{k-x}{\Delta} \leq \binom{k-x}{\Delta'}\binom{x}{a'}$ *is true.*

- *Since* $\rho(a + \Delta,\, x - 2a) < \rho(a + \Delta,\, k - x - 2\Delta)$*, the following statements must be true:*

$$\begin{aligned}
&\rho(a + \Delta,\, x - 2a) = \rho(a' + \Delta',\, k - x - 2\Delta') = 0, \text{ and} \\
&\rho(a' + \Delta',\, x - 2a') = \rho(a + \Delta,\, k - x - 2\Delta) = 1.
\end{aligned}$$

Hence, Inequality 4.39 can be simplified to:

$$\binom{x}{a'}\binom{k-x}{\Delta'} \geq \binom{k-x}{\Delta}\binom{x}{a},$$

which was shown to be true. Therefore, S5 holds.

S6. It is shown that for any two natural numbers a and Δ such that $\frac{x}{2} \leq a \leq x$, $\Delta \leq (k-x)$, $\Delta - a < \frac{k-2x}{2}$, and $\Delta + a > \frac{k}{2}$, if $\rho(a+\Delta,\, x-2a) < \rho(a+\Delta,\, k-x-2\Delta)$, there exists two natural numbers a' and Δ' such that

$$0 \leq a' = \Delta - \frac{k-2x}{2} \leq x \qquad \text{and} \qquad 0 \leq \Delta' = a + \frac{k-2x}{2} \leq k-x,$$

and the following property holds:

$$\begin{aligned}&\binom{x}{a}\binom{k-x}{\Delta}\rho(a+\Delta,\, x-2a) + \binom{x}{a'}\binom{k-x}{\Delta'}\rho(a'+\Delta',\, x-2a') \geq \\ &\binom{k-x}{\Delta}\binom{x}{a}\rho(a+\Delta,\, k-x-2\Delta) + \binom{k-x}{\Delta'}\binom{x}{a'}\rho(a'+\Delta',\, k-x-2\Delta'). \quad (4.44)\end{aligned}$$

The statement is proven by showing that the following statements hold under the conditions given in S6:

- $0 \leq a' \leq x$,
- $0 \leq \Delta' \leq k-x$,
- $\rho(a+\Delta,\, x-2a) = \rho(a'+\Delta',\, k-x-2\Delta')$,
- $\rho(a'+\Delta',\, x-2a') = \rho(a+\Delta,\, k-x-2\Delta)$, *and*
- $\binom{x}{a}\binom{k-x}{\Delta} \leq \binom{k-x}{\Delta'}\binom{x}{a'}$.

The proof steps are as follows:

- $a' \geq 0$ *holds because*

$$\begin{aligned}\Delta + a &> \frac{k}{2} && \text{cf., S6}\\ \Delta + x &> \frac{k}{2} && \text{since } a \leq x\\ \Delta - \frac{k}{2} + x &> 0\\ \Delta - \frac{k-2x}{2} &> 0\\ a' &> 0. && \text{cf., S6}\end{aligned}$$

- $a' \leq x$ *holds because*

$$\begin{aligned} \Delta - a &< \frac{k-2x}{2} && \textit{cf., S6} \\ \Delta - \frac{k-2x}{2} &< a \\ \Delta - \frac{k-2x}{2} &< x && \textit{since } a \leq x \\ a' &< x. \end{aligned}$$

- $\Delta' \geq 0$ *holds because*

$$\begin{aligned} \Delta - a &< \frac{k-2x}{2} && \textit{cf., S6} \\ \Delta &< a + \frac{k-2x}{2} \\ 0 &< a + \frac{k-2x}{2} && \textit{since } \Delta \geq 0 \\ 0 &< \Delta'. \end{aligned}$$

- $\Delta' \leq k - x$ *holds because*

$$\begin{aligned} a \leq x &\leq \frac{k}{2} && \textit{cf., S6 and Lemma 5} \\ a &\leq \frac{k}{2} \\ a + \frac{k}{2} - x &\leq \frac{k}{2} + \frac{k}{2} - x \\ a + \frac{k-2x}{2} &\leq k - x \\ \Delta' &\leq k - x. \end{aligned}$$

- $\rho(a+\Delta,\, x-2a) = \rho(a'+\Delta',\, k-x-2\Delta')$ *and* $\rho(a'+\Delta',\, x-2a') = \rho(a+\Delta,\, k-x-2\Delta)$ *hold for the same reasons discussed in S5 (i.e., the proofs are independent of the conditions in S6).*
- *Let* $M = a - a' = \Delta' - \Delta > 0$ *and* $N = \Delta - a' = \Delta' - a = \frac{k-2x}{2} \geq 0$ *(conditions trivially follow from earlier proof steps). Then,* $\binom{x}{a}\binom{k-x}{\Delta} \leq \binom{k-x}{\Delta'}\binom{x}{a'}$ *is true, as*

shown below:

$$\binom{x}{a}\binom{k-x}{\Delta} = \frac{x!}{a!(x-a)!} \times \frac{(k-x)!}{(a+N-M)!(k-x-a-N+M)!}$$

and

$$\binom{x}{a'}\binom{k-x}{\Delta'} = \frac{x!}{(a-M)!(x-a+M)!} \times \frac{(k-x)!}{(a+N)!(k-x-a-N)!}.$$

Since the numerators are the same (and positive) in both of the equations above, $\binom{x}{a}\binom{k-x}{\Delta} \leq \binom{k-x}{\Delta'}\binom{x}{a'}$ *iff*

$$\frac{a!(x-a)!}{(a-M)!(x-a+M)!} \times \frac{(a+N-M)!(k-x-a-N+M)!}{(a+N)!(k-x-a-N)!} \geq 1.$$

By expanding the factorials, one obtains

$$\frac{\overbrace{(a) \times \cdots \times (a-M+1)}^{M \text{ times}}}{\underbrace{(x-a+M) \times \cdots \times (x-a+1)}_{M \text{ times}}} \times \frac{\overbrace{(k-x-a-N+M) \times \cdots \times (k-x-a-N+1)}^{M \text{ times}}}{\underbrace{(a+N) \times \cdots \times (a+N-M+1)}_{M \text{ times}}}$$

The inequality above can be simplified further, as shown below:

$$\prod_{i=1}^{M} \frac{(a-M+i)}{(x-a+i)} \times \frac{(k-x-a-N+M+1-i)}{(a+N+1-i)} \geq 1. \tag{4.45}$$

Note that

$$\begin{aligned}
(\cancel{a} - \cancel{M} + \cancel{i}) + (k - x - \cancel{a} - N + \cancel{M} + 1 - \cancel{i}) &\stackrel{?}{=} \frac{k}{2} + 1 \\
k - x - N + 1 &\stackrel{?}{=} \frac{k}{2} + 1 \\
k - \cancel{x} - \frac{k}{2} + \cancel{x} + 1 &\stackrel{?}{=} \frac{k}{2} + 1 \\
\frac{k}{2} + 1 &= \frac{k}{2} + 1.
\end{aligned}$$

and

$$\begin{aligned}
(x - \cancel{a} + \cancel{i}) + (\cancel{a} + N + 1 - \cancel{i}) &\stackrel{?}{=} \frac{k}{2} + 1 \\
x + N + 1 &\stackrel{?}{=} \frac{k}{2} + 1 \\
\cancel{x} + \frac{k}{2} - \cancel{x} + 1 &\stackrel{?}{=} \frac{k}{2} + 1 \\
\frac{k}{2} + 1 &= \frac{k}{2} + 1.
\end{aligned}$$

Therefore, Inequality 4.45 can be restated as:

$$\prod_{i=1}^{M} \frac{(a - M + i)}{(x - a + i)} \times \frac{\left(\frac{k}{2} + 1\right) - \left(a - M + i\right)}{\left(\frac{k}{2} + 1\right) - \left(x - a + i\right)} \geq 1. \tag{4.46}$$

Note that Inequality 4.46 holds iff

$$\begin{aligned}
\Big(\big(\tfrac{k}{2}+1\big) - \big((x-a+1)+(x-a+M)\big)\Big)^2 &\geq \Big(\big(\tfrac{k}{2}+1\big) - \big((a-M+1)+(a-\cancel{M}+\cancel{M})\big)\Big)^2 \\
\Big(\big(\tfrac{k}{2}+\cancel{1}\big) - \big(2x-2a+M+\cancel{1}\big)\Big)^2 &\geq \Big(\big(\tfrac{k}{2}+\cancel{1}\big) - \big(2a-M+\cancel{1}\big)\Big)^2 \\
\Big(\frac{k}{2} - 2x + 2a - M\Big)^2 &\geq \Big(\frac{k}{2} - 2a + M\Big)^2.
\end{aligned} \tag{4.47}$$

Since $M = N - (\Delta - a) = \frac{k}{2} - x - \Delta + a$,

$$\begin{aligned}\Big(\frac{k}{2} - 2x + 2a - M\Big)^2 &= \Big(\frac{k}{2} - 2x + 2a - \frac{k}{2} + x + \Delta - a\Big)^2 \\ &= \Big(a - x + \Delta\Big)^2\end{aligned}$$

and

$$\begin{aligned}\Big(\frac{k}{2} - 2a + M\Big)^2 &= \Big(\frac{k}{2} - 2a + \frac{k}{2} - x - \Delta + a\Big)^2 \\ &= \Big(k - a - x - \Delta\Big)^2.\end{aligned}$$

Therefore, Inequality 4.47 can be simplified to:

$$\Big(a - x + \Delta\Big)^2 \geq \Big(k - a - x - \Delta\Big)^2 \tag{4.48}$$

or to

$$\Big(\Delta + a - x\Big)^2 \geq \Big(k - x - (\Delta + a)\Big)^2. \tag{4.49}$$

Note that $\Delta + a > \frac{k}{2}$ *and* $\frac{k}{2} \geq x$*, therefore,* $\Delta + a - x > 0$*, and there are only two cases to consider:*

- *Case I:* $(k - x) - (\Delta + a) \geq 0$*, and*
- *Case II:* $(k - x) - (\Delta + a) < 0$.

For Case I, it needs to be shown that

$$\begin{aligned}(\Delta + a) - x &\overset{?}{\geq} (k - x) - (\Delta + a) \\ 2(\Delta + a) &\overset{?}{\geq} k - 2x \\ \Delta + a &\overset{?}{\geq} \frac{k}{2} - x\end{aligned}$$

Since $\Delta + a > \frac{k}{2}$ *and* $x \geq 0$,

$$\Delta + a \geq \frac{k}{2} - x.$$

For Case II, it needs to be shown that

$$\begin{aligned} \cancel{(\Delta + a)} - x &\overset{?}{\geq} \cancel{(\Delta + a)} - (k - x) \\ -x &\overset{?}{\geq} -k + x \\ k &\overset{?}{\geq} 2x, \end{aligned}$$

which is true by definition. Therefore, Inequalities 4.40–4.43 must be true. Consequently, $\binom{x}{a}\binom{k-x}{\Delta} \leq \binom{k-x}{\Delta'}\binom{x}{a'}$ *is true.*

- *Since* $\rho(a + \Delta,\, x - 2a) < \rho(a + \Delta,\, k - x - 2\Delta)$, *the following statements must be true:*

$$\begin{aligned} &\rho(a + \Delta,\, x - 2a) = \rho(a' + \Delta',\, k - x - 2\Delta') = 0, \text{ and} \\ &\rho(a' + \Delta',\, x - 2a') = \rho(a + \Delta,\, k - x - 2\Delta) = 1. \end{aligned}$$

Hence, Inequality 4.44 can be simplified to:

$$\binom{x}{a'}\binom{k-x}{\Delta'} \geq \binom{k-x}{\Delta}\binom{x}{a},$$

which was shown to be true. Therefore, S6 holds.

S7. According to Equation 4.7, the probabilities $\text{PR}_{==}(\vec{r}_1,\, \vec{r}_2,\, x)$ *and* $\text{PR}_{==}(\vec{r}_3,\, \vec{r}_4,\, k-x)$ *can be expanded as the summation of products that are shown in Table 4.4. For brevity, the constant multiplier*

$$\frac{\binom{k}{k-x}}{2^{2k}}$$

has been omitted from both equations.

$\mathrm{PR}_{==}(\vec{r}_1,\,\vec{r}_2,\,x)$				$\mathrm{PR}_{==}(\vec{r}_3,\,\vec{r}_4,\,k-x)$			
$\mathbf{a}$	$\boldsymbol{\Delta}$			$\mathbf{a'}$	$\boldsymbol{\Delta'}$		
0	0	$\binom{x}{0}\binom{k-x}{0}$	$\rho(0,x)$	0	0	$\binom{k-x}{0}\binom{x}{0}$	$\rho(0,k-x)$
0	1	$\binom{x}{0}\binom{k-x}{1}$	$\rho(1,x)$	1	0	$\binom{k-x}{1}\binom{x}{0}$	$\rho(1,k-x-2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
0	$k-x-1$	$\binom{x}{0}\binom{k-x}{k-x-1}$	$\rho(k-x-1,x)$	$k-x-1$	0	$\binom{k-x}{k-x-1}\binom{x}{0}$	$\rho(k-x-1,-k+x+2)$
0	$k-x$	$\binom{x}{0}\binom{k-x}{k-x}$	$\rho(k-x,x)$	$k-x$	0	$\binom{k-x}{k-x}\binom{x}{0}$	$\rho(k-x,-k+x)$
1	0	$\binom{x}{1}\binom{k-x}{0}$	$\rho(1,x-2)$	0	1	$\binom{k-x}{0}\binom{x}{1}$	$\rho(1,k-x)$
1	1	$\binom{x}{1}\binom{k-x}{1}$	$\rho(2,x-2)$	1	1	$\binom{k-x}{1}\binom{x}{1}$	$\rho(2,k-x-2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	$k-x-1$	$\binom{x}{1}\binom{k-x}{k-x-1}$	$\rho(k-x,x-2)$	$k-x-1$	1	$\binom{k-x}{k-x-1}\binom{x}{1}$	$\rho(k-x,-k+x+2)$
1	$k-x$	$\binom{x}{1}\binom{k-x}{k-x}$	$\rho(k-x+1,x-2)$	$k-x$	1	$\binom{k-x}{k-x}\binom{x}{1}$	$\rho(k-x+1,-k+x)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$x-1$	0	$\binom{x}{x-1}\binom{k-x}{0}$	$\rho(x-1,-x+2)$	0	$x-1$	$\binom{k-x}{0}\binom{x}{x-1}$	$\rho(x-1,k-x)$
$x-1$	1	$\binom{x}{x-1}\binom{k-x}{1}$	$\rho(x,-x+2)$	1	$x-1$	$\binom{k-x}{1}\binom{x}{x-1}$	$\rho(x,k-x-2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$x-1$	$k-x-1$	$\binom{x}{x-1}\binom{k-x}{k-x-1}$	$\rho(k-2,-x+2)$	$k-x-1$	$x-1$	$\binom{k-x}{k-x-1}\binom{x}{x-1}$	$\rho(k-2,-k+x+2)$
$x-1$	$k-x$	$\binom{x}{x-1}\binom{k-x}{k-x}$	$\rho(k-1,-x+2)$	$k-x$	$x-1$	$\binom{k-x}{k-x}\binom{x}{x-1}$	$\rho(k-1,-k+x)$
x	0	$\binom{x}{x}\binom{k-x}{0}$	$\rho(x,-x)$	0	x	$\binom{k-x}{0}\binom{x}{x}$	$\rho(x,k-x)$
x	1	$\binom{x}{x}\binom{k-x}{1}$	$\rho(x+1,-x)$	1	x	$\binom{k-x}{1}\binom{x}{x}$	$\rho(x+1,k-x-2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x	$k-x-1$	$\binom{x}{x}\binom{k-x}{k-x-1}$	$\rho(k-1,-x)$	$k-x-1$	x	$\binom{k-x}{k-x-1}\binom{x}{x}$	$\rho(k-1,-k+x+2)$
x	$k-x$	$\binom{x}{x}\binom{k-x}{k-x}$	$\rho(k,-x)$	$k-x$	x	$\binom{k-x}{k-x}\binom{x}{x}$	$\rho(k,-k+x)$

Table 4.4: Pairings of products

S8. Each term on the left-hand side of Table 4.4 can be paired up with a term on the right-hand side as follows:

- *Pair-up* $\binom{x}{a}\binom{k-x}{\Delta}\rho(a{+}\Delta,\,x{-}2a)$ *with* $\binom{k-x}{a'}\binom{x}{\Delta'}\rho(a'{+}\Delta',\,k{-}x{-}2a')$, *where* $a'=\Delta$ *and* $\Delta'=a$ *if*

$$\Big(0\le a\le\frac{x}{2},\qquad \Delta\le(k-x)\ \textit{and}\qquad \Delta-a\le\frac{k-2x}{2}\Big)\ \textit{or}$$
$$\Big(\frac{x}{2}\le a\le x,\qquad \Delta\le(k-x)\ \textit{and}\qquad \Delta-a\ge\frac{k-2x}{2}\Big).$$

- *Pair-up* $\binom{x}{a}\binom{k-x}{\Delta}\rho(a{+}\Delta,\, x{-}2a)$ *with* $\binom{k-x}{a'}\binom{x}{\Delta'}\rho(a'{+}\Delta',\, k{-}x{-}2a')$, *where* $a' = k{-}x{-}\Delta$ *and* $\Delta' = x{-}a$ *if*

$$\big(0 \le a \le \frac{x}{2}, \qquad \Delta \le (k-x) \text{ and } \qquad \Delta + a \ge \frac{k}{2}\big) \text{ or}$$
$$\big(\frac{x}{2} \le a \le x, \qquad \Delta \le (k-x) \text{ and } \qquad \Delta + a \le \frac{k}{2}\big).$$

- *Else, pair-up* $\binom{x}{a}\binom{k-x}{\Delta}\rho(a{+}\Delta,\, x{-}2a)$ *with* $\binom{k-x}{a'}\binom{x}{\Delta'}\rho(a'{+}\Delta',\, k{-}x{-}2a')$, *where* $a' = \Delta$ *and* $\Delta' = a$.

S9. Based on the pairings outlined in S8, it can be shown that the summation of products on the left-hand side of Table 4.4 is always greater than or equal to the summation of products on the right-hand side. The reason is as follows: For the first case in S8, statements S1 and S2 suggest that the product on the left-hand side of a pairing is greater than or equal to the product on the right-hand side. For the second case in S8, the same conclusion can be reached using statements S3 and S4. Whether the product on the left-hand side of a pairing in the third case in S8 is greater than or equal to the product on the right-hand side is questionable. If it is greater than or equal to the product on the right-hand side, then there is no problem. Otherwise, it needs to be checked whether the difference in the products (which is a loss for the summation on the left-hand side) is compensated elsewhere. Fortunately, statements S5 and S6 suggest that there exists another pairing (already covered by the first four cases) where the difference in products compensates the loss in the summation of the left-hand side. It can be easily inferred from the conditions of S5 and S6 that for each questionable (and problematic) case, the pairing that compensates the loss is a distinct one. Therefore, the summation on the left-hand side of Table 4.4 must be greater than or equal to the summation on the right-hand side, thus, proving Lemma 5. □

Lemma 6. *Let* $\delta^M(\vec{r}_i,\, \vec{r}_j)$ *denote the Manhattan distance between any two record utilization vectors* $\vec{r}_i$ *and* $\vec{r}_j$, *and let* m *denote the number of rightmost bits that are dropped by* h_2, *which is utilized in* h *(cf., Definition 10). Furthermore, let* $\text{PR}_{==}\big(\vec{r}_i,\, \vec{r}_j,\, x\big)$ *denote the posterior probability that, for any two record utilization vectors* $\vec{r}_i$ *and* $\vec{r}_j$, $h(\vec{r}_i) = h(\vec{r}_j)$ *provided that* $\delta^M(\vec{r}_i,\, \vec{r}_j) = x$. *Given any four record utilization vectors* $\vec{r}_1$, $\vec{r}_2$, $\vec{r}_3$ *and* $\vec{r}_4$ *with size* λb *such that* $\delta^M(\vec{r}_1,\, \vec{r}_2) = x$ *and* $\delta^M(\vec{r}_3,\, \vec{r}_4) = \lambda b - x$, *the following property holds for any* $x \le \frac{\lambda}{2}$ *and* $m = \Upsilon b$*:*

$$\text{PR}_{==}\big(\vec{r}_1,\, \vec{r}_2,\, x\big) \ge \text{PR}_{==}\big(\vec{r}_3,\, \vec{r}_4,\, \lambda b - x\big) \tag{4.50}$$

where $h = h_2 \circ h_1$ (cf., Definition 10), $b \in \mathbb{Z}_{1 \cdots \infty}$ denotes the number of entries in the record utilization counters produced by h_1, λ is an even and positive number, and $\Upsilon \in \mathbb{Z}_{1 \cdots \lambda-1}$.

Proof 6. *Lemma 6 is proven by induction on b.*

***Base case**: It needs to be shown that Equation 4.50 holds when $b = 1$. The proof trivially follows from Lemma 5.*

***Inductive Step**: Assuming that Lemma 6 holds for $b \leq \alpha$ where α is a natural number greater than or equal to 1, it needs to be shown that it also holds for $b = \alpha + 1$. The proof steps are as follows:*

S1. Let $\vec{r}_i[j]$ denote the group of bits in a record utilization vector r_i that have the same hash value j with respect to the hash function f, which is utilized within h_1 (cf., Definition 4.50). (Assume that within each $\vec{r}_i[j]$, the ordering of bits in $\vec{r}_i$ is preserved.)

S2. Recall that the Manhattan distance between any two record utilization vectors is the summation of their individual Manhattan distances within each group of bits that share the same hash value with respect to f (cf., Equation 4.4). Therefore, since $\delta^M(\vec{r}_1, \vec{r}_2) = x$,

$$\begin{aligned} &\text{if } \delta^M(\vec{r}_1[\alpha], \vec{r}_2[\alpha]) = a, \text{ then} \\ &\quad \delta^M(\vec{r}_1[0] \cdots \vec{r}_1[\alpha-1], \vec{r}_2[0] \cdots \vec{r}_2[\alpha-1]) = x - a \end{aligned}$$

where $\vec{r}_i[0] \cdots \vec{r}_i[\alpha-1]$ denotes the concatenation of the corresponding bit vectors.

S3. Also note that $h(\vec{r}_1) = h(\vec{r}_2)$ if and only if $h(\vec{r}_1[j]) = h(\vec{r}_2[j])$ for all $j \in \mathbb{Z}_{0 \cdots \alpha}$. The reason is that $h = h_2 \circ h_1$ and $h(\vec{r}_1)$ (respectively, $h(\vec{r}_2)$) corresponds to the bit vector that is produced by interleaving the bits in the binary representations of $h_1(\vec{r}_1[0])$ $\ldots$ $h_1(\vec{r}_1[\alpha])$ (respectively, $h_1(\vec{r}_2[0])$ $\ldots$ $h_1(\vec{r}_2[\alpha])$) and cutting off the rightmost Υb bits [144]. Consequently, for $h(\vec{r}_1) = h(\vec{r}_2)$ to be true, for all $j \in \mathbb{Z}_{0 \cdots \alpha}$, $h_1(\vec{r}_1[j])$ and $h_1(\vec{r}_2[j])$ must have the same sequence of bits except for the rightmost Υ, which means that $h(\vec{r}_1[j])$ and $h(\vec{r}_2[j])$.

S4. As a consequence of statements S2 and S3, the following property holds:

$$\text{PR}_{==}(\vec{r}_1, \vec{r}_2, x) = \sum_{a=0}^{x} \text{PR}_{==}(\vec{r}_1[0] \cdots \vec{r}_1[\alpha-1], \vec{r}_2[0] \cdots \vec{r}_2[\alpha-1], x-a) \times \text{PR}_{==}(\vec{r}_1[\alpha], \vec{r}_2[\alpha], a) \times \text{PR}_a$$

where $\text{PR}_a \in \{\text{PR}_0, \ldots, \text{PR}_x\}$ *denotes a constant that represents the probability that* $\delta^M(\vec{r}_1[\alpha], \vec{r}_2[\alpha]) = a$ *among all possible configurations such that* $\delta^M(\vec{r}_1, \vec{r}_2) = x$.

S5. Statement S2 holds also for $\vec{r}_3$ *and* $\vec{r}_4$*; therefore, since* $\delta^M(\vec{r}_3, \vec{r}_4) = \lambda b - x$,

$$\begin{aligned} &\textit{if } \delta^M(\vec{r}_3[\alpha], \vec{r}_4[\alpha]) = a', \textit{ then} \\ &\quad \delta^M(\vec{r}_3[0] \cdots \vec{r}_3[\alpha-1], \vec{r}_4[0] \cdots \vec{r}_4[\alpha-1]) = \lambda b - x - a' \end{aligned}$$

where $\vec{r}_i[0] \cdots \vec{r}_i[\alpha-1]$ *denotes the concatenation of the corresponding bit vectors.*

S6. Since there are $\lambda b - \lambda$ *bits in* $\vec{r}_3[0] \cdots \vec{r}_3[\alpha-1]$ *and* $\vec{r}_4[0] \cdots \vec{r}_4[\alpha-1]$*, the edit distance between these two bit vectors can be at most* $\lambda b - \lambda$*, thus,*

$$\delta^M(\vec{r}_3[0] \cdots \vec{r}_3[\alpha-1], \vec{r}_4[0] \cdots \vec{r}_4[\alpha-1]) \leq \lambda b - \lambda.$$

Therefore, $(\lambda - x) \leq a' \leq \lambda$ *must hold.*

S7. Consequently, similar to S4, the following statement must be true:

$$\begin{aligned} &\text{PR}_{==}\big(\vec{r}_3, \vec{r}_4, \lambda b - x\big) = \\ &\quad \sum_{a'=\lambda-x}^{\lambda} \begin{array}{l} \text{PR}_{==}\big(\vec{r}_3[0] \cdots \vec{r}_3[\alpha-1], \vec{r}_4[0] \cdots \vec{r}_4[\alpha-1], \lambda b - x - a'\big) \times \\ \text{PR}_{==}\big(\vec{r}_3[\alpha], \vec{r}_4[\alpha], a'\big) \times \text{PR}_{a'} \end{array} \end{aligned}$$

where $\text{PR}_{a'} \in \{\text{PR}_{\lambda-x}, \ldots, \text{PR}_\lambda\}$ *denotes a constant that represents the probability that* $\delta^M(\vec{r}_3[\alpha], \vec{r}_4[\alpha]) = a'$ *among all possible configurations such that* $\delta^M(\vec{r}_3, \vec{r}_4) = \lambda b - x$.

S8. The possible configurations in the summations in S4 and S7 for $\text{PR}_{==}\big(\vec{r}_1, \vec{r}_2, x\big)$ *and* $\text{PR}_{==}\big(\vec{r}_3, \vec{r}_4, \lambda b - x\big)$ *can be paired up as shown in Table 4.5.*

a		**a′**	
0	$\text{PR}_0 \times$ $\text{PR}_{==}\big(\vec{r}_1[\alpha],\, \vec{r}_2[\alpha],\, 0\big) \times$ $\text{PR}_{==}\big(\vec{r}_1[0]\cdots\vec{r}_1[\alpha-1],\, \vec{r}_2[0]\cdots\vec{r}_2[\alpha-1],\, x\big)$	λ	PR'_{λ} $\text{PR}_{==}\big(\vec{r}_3[\alpha],\, \vec{r}_4[\alpha],\, \lambda\big) \times$ $\text{PR}_{==}\big(\vec{r}_3[0]\cdots\vec{r}_3[\alpha-1],\, \vec{r}_4[0]\cdots\vec{r}_4[\alpha-1],\, \lambda b - x - \lambda\big)$
1	$\text{PR}_1 \times$ $\text{PR}_{==}\big(\vec{r}_1[\alpha],\, \vec{r}_2[\alpha],\, 1\big) \times$ $\text{PR}_{==}\big(\vec{r}_1[0]\cdots\vec{r}_1[\alpha-1],\, \vec{r}_2[0]\cdots\vec{r}_2[\alpha-1],\, x-1\big)$	$\lambda-1$	$\text{PR}'_{\lambda-1} \times$ $\text{PR}_{==}\big(\vec{r}_3[\alpha],\, \vec{r}_4[\alpha],\, \lambda-1\big) \times$ $\text{PR}_{==}\big(\vec{r}_3[0]\cdots\vec{r}_3[\alpha-1],\, \vec{r}_4[0]\cdots\vec{r}_4[\alpha-1],\, \lambda b - x - \lambda + 1\big)$
$\vdots$		$\vdots$	
x	$\text{PR}_x \times$ $\text{PR}_{==}\big(\vec{r}_1[\alpha],\, \vec{r}_2[\alpha],\, x\big) \times$ $\text{PR}_{==}\big(\vec{r}_1[0]\cdots\vec{r}_1[\alpha-1],\, \vec{r}_2[0]\cdots\vec{r}_2[\alpha-1],\, 0\big)$	$\lambda-x$	$\text{PR}'_{\lambda-x} \times$ $\text{PR}_{==}\big(\vec{r}_3[\alpha],\, \vec{r}_4[\alpha],\, \lambda-x\big) \times$ $\text{PR}_{==}\big(\vec{r}_3[0]\cdots\vec{r}_3[\alpha-1],\, \vec{r}_4[0]\cdots\vec{r}_4[\alpha-1],\, \lambda b - \lambda\big)$

Table 4.5: Possible pairs of configurations

S9. For each pair of configurations in Table 4.5, (i) $a \leq x \leq \frac{\lambda}{2} \leq \lambda - a$ and (ii) $a+\lambda-a = \lambda$ hold. Therefore, Lemma 5 suggests that

$$\text{PR}_{==}\big(\vec{r}_1[\alpha],\, \vec{r}_2[\alpha],\, a\big) \geq \text{PR}_{==}\big(\vec{r}_3[\alpha],\, \vec{r}_4[\alpha],\, \lambda - a\big).$$

S10. For each pair of configurations in Table 4.5, $(x-a) \leq \frac{\lambda(b-1)}{2} \leq \lambda b - x - \lambda + a$ holds:

- *$(x-a) \leq \frac{\lambda(b-1)}{2}$ is true because*
  - *$x \leq \frac{\lambda}{2}$ by definition (cf., Lemma 6);*
  - *$a \geq 0$, therefore, $(x-a) \leq \frac{\lambda}{2}$; and*
  - *$\frac{\lambda}{2} \leq \frac{\lambda(b-1)}{2}$ for $b \geq 2$, which is the case for b in the inductive step.*
- *$\lambda b - x - \lambda + a \geq \frac{\lambda(b-1)}{2}$ is true because*
  - *$x \leq \frac{\lambda}{2}$ (cf., Lemma 6) and $a \geq 0$, therefore,*

$$\begin{aligned} \lambda b - \lambda - x + a &\geq \lambda b - \lambda - x \\ &\geq \lambda b - \lambda - \frac{\lambda}{2} \\ &\geq \lambda b - \frac{3\lambda}{2}. \end{aligned}$$

  - *For $b \geq 2$, which is true for the inductive step, $\lambda b - \frac{3\lambda}{2} \geq \frac{\lambda(b-1)}{2}$ must also hold, therefore $\lambda b - x - \lambda + a \geq \frac{\lambda(b-1)}{2}$ is true.*

Consequently, according to the inductive assumption, the following statement must be true:

$$\begin{aligned}&\text{PR}_{==}\big(\vec{r}_1[0]\cdots\vec{r}_1[\alpha-1],\,\vec{r}_2[0]\cdots\vec{r}_2[\alpha-1],\,x-a\big)\\&\quad\geq\text{PR}_{==}\big(\vec{r}_3[0]\cdots\vec{r}_3[\alpha-1],\,\vec{r}_4[0]\cdots\vec{r}_4[\alpha-1],\,\lambda b-x-\lambda+a\big).\end{aligned}$$

S11. For each pair of configurations in Table 4.5, $\text{PR}_a = \text{PR}'_{\lambda-a}$ *because*

$$\text{PR}_a=\frac{\binom{\lambda}{\lambda-x}}{2^\lambda},\quad \text{PR}'_{\lambda-a}=\frac{\binom{\lambda}{\lambda-\lambda+a}}{2^\lambda}=\frac{\binom{\lambda}{a}}{2^\lambda}\quad\textit{and}\quad\binom{\lambda}{\lambda-x}=\binom{\lambda}{x}.$$

S12. According to statements S9, S10 and S11, for each pair of configurations in Table 4.5, the product on the left hand side is always greater than or equal to the product on the right hand side, which means:

$$\text{PR}_{==}\big(\vec{r}_1,\,\vec{r}_2,\,x\big)\geq\text{PR}_{==}\big(\vec{r}_3,\,\vec{r}_4,\,\lambda b-x\big).$$

Thus, Lemma 4.50 holds also for $b=\alpha+1$ *Consequently, Lemma 4.50 is proven by induction on* b. □

Theorem 7. *Let* $\delta^M(\vec{r}_i,\,\vec{r}_j)$ *denote the Manhattan distance between any two record utilization vectors* $\vec{r}_i$ *and* $\vec{r}_j$ *with size* λb*, and let* m *denote the number of rightmost bits that are dropped by* h_2*, which is utilized in* h *(cf., Definition 10). For any* $\delta^M\leq\frac{\lambda}{2}$*,* h *is* $(\delta^M,\,\lambda b-\delta^M,\,\text{PR}_1,\,\text{PR}_2)$*-sensitive for some* $\text{PR}_1\geq\text{PR}_2$ *where* $h=h_2\circ h_1$ *(cf., Definition 10),* λ *is an even and positive number,* $b\in\mathbb{Z}_{1\,\ldots\,\infty}$ *denotes the number of entries in the record utilization counters produced by* h_1*,* $m=\Upsilon b$*, and* $\Upsilon\in\mathbb{Z}_{1\,\ldots\,\lambda-1}$.

Proof 7. *The proof steps are as follows:*

S1. PR_1 *corresponds to the posterior probability that* $h(\vec{r}_i)=h(\vec{r}_j)$ *for any two record utilization vectors* $\vec{r}_i$ *and* $\vec{r}_j$ *with size* λb *such that* $\delta^M(\vec{r}_i,\vec{r}_j)\leq\delta^M$ *[180].*

S2. PR_2 *corresponds to the posterior probability that* $h(\vec{r}_i)\neq h(\vec{r}_j)$ *for any two record utilization vectors* $\vec{r}_i$ *and* $\vec{r}_j$ *with size* λb *such that* $\delta^M(\vec{r}_i,\vec{r}_j)\geq\lambda b-\delta^M$ *[180].*

S3. Note that

$$\mathrm{PR}_1 = \frac{\sum_{x=0}^{\delta^M} \mathrm{PR}_{==}\left(\vec{r}_i,\, \vec{r}_j,\, x\right) \binom{\lambda b}{\lambda b - x}}{2^{\lambda b}}$$

$$\mathrm{PR}_2 = \frac{\sum_{x=0}^{\delta^M} \mathrm{PR}_{==}\left(\vec{r}_i,\, \vec{r}_j,\, \lambda b - x\right) \binom{\lambda b}{x}}{2^{\lambda b}}.$$

S4. For all $x \in \mathbb{Z}_{0 \ldots \delta^M}$ $\binom{\lambda b}{\lambda b - x} = \binom{\lambda b}{x}$.

S5. Therefore, for all $x \in \mathbb{Z}_{0 \ldots \delta^M}$

$$\frac{\binom{\lambda b}{\lambda b - x}}{2^{\lambda b}} = \frac{\binom{\lambda b}{x}}{2^{\lambda b}}.$$

S6. According to Lemma 6, $\mathrm{PR}_{==}\left(\vec{r}_i,\, \vec{r}_j,\, x\right) \geq \mathrm{PR}_{==}\left(\vec{r}_i,\, \vec{r}_j,\, \lambda b - x\right)$. *Consequently,* $\mathrm{PR}_1 \geq \mathrm{PR}_2$. □

Theorem 7 suggests that h is a function with locality-sensitive properties, and can be used to approximate the clustering problem. However, it must be noted that the sensitivity analysis of h is conservative. In other words, it is believed that stronger statements can be made about h, in particular, due to the empirical observation that $\mathrm{PR}_{==}\left(\vec{r}_i,\, \vec{r}_j,\, x\right)$ is a monotonically decreasing function. Proving this conjecture is left as future work.

4.3.4 Achieving and Maintaining Tighter Bounds on Tunable-LSH

Next, techniques are presented for reducing the approximation error of h_1, hence h. First, the *load factor* of an entry of a record utilization counter is defined.

Definition 11 (Load Factor)**.** *Given a record utilization counter* $\vec{c} = (c[0], \ldots, c[b{-}1])$ *with size* b*, the* load factor *of the* i^{th} *entry is* $c[i]$. □

Theorem 8 (Effects of Grouping)**.** *Given two record utilization vectors* $\vec{r_1}$ *and* $\vec{r_2}$ *with size* k*, let* $\vec{c_1}$ *and* $\vec{c_2}$ *denote two record utilization counters with size* $b = 1$ *such that* $\vec{c_1} = h_1(\vec{r_1})$

and $\vec{c_2} = h_1(\vec{r_2})$. *Then,*

$$\Pr\left(\begin{array}{l|l} \delta^M(\vec{c_1},\vec{c_2}) & c_1[0] = l_1 \text{ AND} \\ \quad = \delta^M(\vec{r_1},\vec{r_2}) & c_2[0] = l_2 \end{array}\right) = \gamma \tag{4.51}$$

where

$$\gamma = \frac{\binom{l_{max}}{l_{min}}\binom{k}{l_{max}}}{\binom{k}{l_{max}}\binom{k}{l_{min}}} \tag{4.52}$$

and

$$l_{max} = max(l_1, l_2)$$
$$l_{min} = min(l_1, l_2).$$

Proof 8. *Let* $\vec{r}_{max}$ *denote the record utilization vector with the most number of* 1*-bits among* $\vec{r_1}$ *and* $\vec{r_2}$*, and let* $\vec{r}_{min}$ *denote the vector with the least number of* 1*-bits. When* $b = 1$*,* $\delta^M(\vec{c_1},\vec{c_2}) = \delta(\vec{r_1},\vec{r_2})$ *holds if and only if the number of* 1*-bits on which* $\vec{r_1}$ *and* $\vec{r_2}$ *are aligned is* l_{min} *because in that case, both* $\delta^M(\vec{c_1},\vec{c_2})$ *and* $\delta(\vec{r_1},\vec{r_2})$ *are equal to* $l_{max} - l_{min}$ *(note that* $\delta^M(\vec{c_1},\vec{c_2})$ *is always equal to* $l_{max} - l_{min}$*). Assuming that the positions of* 1*-bits in* $\vec{r}_{max}$ *are fixed, there are* $\binom{l_{max}}{l_{min}}$ *possible ways of arranging the* 1*-bits of* $\vec{r}_{min}$ *such that* $\delta(\vec{r_1},\vec{r_2}) = l_{max} - l_{min}$*. Since the* 1*-bits of* $\vec{r}_{max}$ *can be arranged in* $\binom{k}{l_{max}}$ *different ways, there are* $\binom{l_{max}}{l_{min}}\binom{k}{l_{max}}$ *combinations such that* $\delta^M(\vec{c_1},\vec{c_2}) = \delta(\vec{r_1},\vec{r_2})$*. Note that in total, the bits of* $\vec{r_1}$ *and* $\vec{r_2}$ *can be arranged in* $\binom{k}{l_{max}}\binom{k}{l_{min}}$ *possible ways; therefore, Equations 4.51 and 4.52 describe the posterior probability that* $\delta^M(\vec{c_1},\vec{c_2}) = \delta(\vec{r_1},\vec{r_2})$*, given* $c_1[0] = l_1$ *and* $c_2[0] = l_2$*.* □

According to Equations 4.51 and 4.52 in Theorem 8, the probability that $\delta^M(\vec{c_1},\vec{c_2})$ is an approximation of $\delta(\vec{r_1},\vec{r_2})$, but that it is not exactly equal to $\delta(\vec{r_1},\vec{r_2})$ is lower for load factors that are close or equal to zero and likewise for load factors that are close or equal to $\lceil \frac{k}{b} \rceil$ (cf., Fig. 4.8). This property suggests that by carefully choosing f, it is possible to achieve even tighter error bounds for h_1. For $b \geq 2$, the probabilities for each group of bits need to be multiplied as illustrated in the proof of Lemma 6. Therefore, the algorithm for tuning f aims to make sure that the load factors are either low or high for as many of the groups as possible.

Contrast the matrices in Figures 4.7b and 4.7c, which contain the same query access

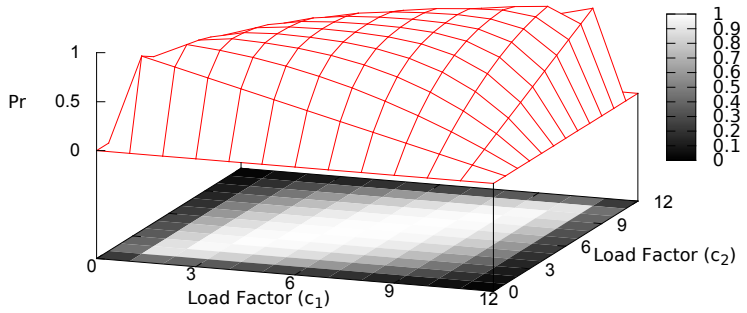


Figure 4.8: $\Pr(\delta^M \neq \delta)$ for $k = 12$, $b = 1$ and across varying load factors

vectors, but the columns are grouped in two different ways[3]: (i) in Figure 4.7b, the grouping is based on the original sequence of execution, and (ii) in Figure 4.7c, queries with similar access patterns are grouped together. Figure 4.7d and Figure 4.7e represent the corresponding record utilization counters for the record utilization vectors in the matrices in Figure 4.7b and Figure 4.7c, respectively. Take $\vec{r}_3$ and $\vec{r}_5$, for instance. Their actual Hamming distance with respect to q_0–q_7 is 8. Now consider the transformed matrices. According to Figure 4.7d, the Hamming distance lower bound is 0, whereas according to Figure 4.7e, it is 8. Clearly, the bounds in the second representation are closer to the original. The reason is as follows. Even though $\vec{r}_3$ and $\vec{r}_5$ differ on all the bits for q_0–q_7, when the bits are grouped as in Figure 4.7b, the counts alone cannot distinguish the two bit vectors. In contrast, if the counts are computed based on the grouping in Figure 4.7c (which clearly places the 1-bits in separate groups), the counts indicate that the two bit vectors are indeed different.

The observations above are in accordance with Theorem 8. Consequently, the following optimization can be made: Instead of randomly choosing a hash function, f can be constructed such that it maps queries with similar access vectors (i.e., columns in the matrix) to the same hash value. This way, it is possible to obtain record utilization counters with entries that have either very high or very low load factors (cf., Definition 10), thus, decreasing the probability of error (cf., Theorem 8).

Consequently, a technique is developed to efficiently determine groups of queries with similar access patterns and to adaptively maintain these groups as the access patterns change. The developed technique consists of two parts: (i) to approximate the similarity between any two queries, the technique relies on the Min-Hash scheme [49], and (ii) to adaptively group similar queries, an incremental version of a multidimensional scaling

[3]Groups are separated by vertical dashed lines.

Symbol	*Description*
begin	natural number between $0 \ldots (k-1)$, initial value is 0
size	natural number between $0 \ldots (k-1)$, keeps track of the number of query access vectors that are currently being maintained, initial value is 0
$H_{k \times ?}$	matrix that contains MIN-HASH values for each query access vector
$S[]$	array of *vector*(s), one for each MDS query point, that pairs each MDS query point with a *random* subset of points
$N[]$	array of *max-heap*(s), one for each MDS query point, that pairs each MDS query point with a set of *neighboring* points
$X[]$	array of *float*(s), represents the coordinate (single dimensional) of each MDS query point
$V[]$	array of *float*(s), represents the current (directional) velocity of each MDS query point

Table 4.6: Data structures referenced in algorithms

(MDS) algorithm [142] has been developed.

MIN-HASH offers a quick and efficient way of approximating the similarity, (more specifically, the Jaccard similarity [117]), between two sets of integers. Therefore, to use it, the query access vectors need to be translated into a set of positional identifiers that correspond to the records for which the bits in the vector are set to 1.[4] For example, according to Figure 4.7a, $\vec{q_1}$ should be represented with the set $\{0, 5, 6\}$ because r_0, r_5 and r_6 are the only records for which the bits are set to 1. Note that, the original query access vectors do not need to be stored at all. In fact, after the access patterns over a query are determined, its MIN-HASH value is computed and only that value is stored. This is important for keeping the memory overhead of the algorithm low.

Queries with similar access patterns are grouped together using a multidimensional scaling (MDS) algorithm [130] that was originally developed for data visualization, and has recently been used for clustering [41]. Given a set of points and a distance function, MDS assigns coordinates to points such that their original distances are preserved as much as possible. In one efficient implementation [142], each point is initially assigned a random set of coordinates, but these coordinates are adjusted iteratively based on a spring-force analogy. That is, it is assumed that points exert a force on each other that is proportional

[4] In practice, this translation never takes place because chameleon-db maintains positional vectors to begin with.

to the difference between their actual and observed distances, where the latter refers to the distance that is computed from the algorithm-assigned coordinates. These forces are used for computing the current velocity (V in Table 4.6) and the approximated coordinates of a point (X in Table 4.6). The intuition is that, after successive iterations, the system will reach equilibrium, at which point, the approximated coordinates can be reported. Since computing all pairwise distances can be prohibitively expensive, the algorithm relies on a combination of sampling ($S[]$ in Table 4.6) and maintaining for each point, a list of its nearest neighbours ($N[]$ in Table 4.6)—only these distances are used in computing the net force acting on a point. Then, the nearest neighbours are updated in each iteration by removing the most distant neighbour of a point and replacing it with a new point from the random sample if the distance between the point and the random sample is smaller than the distance between the point and its most distant neighbour.

The algorithm cannot be used directly for the purposes herein because it is not incremental. Therefore, a revised MDS algorithm is proposed in this section that incorporates the following modifications:

1. Each point in the revised algorithm represents a query access vector. However, since visualizing these points is not the main concern, but rather clustering them is, the algorithm is revised to place these points along a single dimension. Then, by dividing the coordinate space into consecutive regions, it is possible to determine similar query access vectors.

2. Instead of computing the coordinates of all of the points at once, the revised algorithm makes incremental adjustments to the coordinates every time reconfiguration is needed.

The revised algorithm is given in Algorithm 11. First, the algorithm decides which MDS point to assign to the new query access vector $\vec{q_t}$ (line 2). It clears the array and the heap data structures containing, respectively, (i) the randomly sampled, and (ii) the neighbouring set of points (lines 3–4). Furthermore, it assigns a random coordinate to the point within the interval $[-0.5, 0.5]$ (line 5), and resets its velocity to 0 (line 6). Next, it computes the Min-Hash value of $\vec{q_t}$ and stores it in $H[\text{pos}]$ (line 7). Then, it makes two passes over all the points in the system (lines 13–21), while first updating their sample and neighbouring lists (line 15), computing the net forces acting on them based on the Min-Hash distances and updating their velocities (line 16); and then updating their coordinates (line 20).

Algorithm 11 Reconfigure-F

Require:

$\vec{\mathbf{q_t}}$: query access vector produced at time t

Ensure:

Coordinates of MDS points are updated, which are used in determining the outcome of f

```
1:  procedure RECONFIGURE-F(q_t)
2:      pos ← (begin + size) % k
3:      S[pos].clear()
4:      N[pos].clear()
5:      X[pos] ← −0.5 + rand() / RAND-MAX
6:      V[pos] ← 0
7:      H[pos] ← MIN-HASH(q_t)
8:      if size < k then
9:          size += 1
10:     else
11:         begin = (begin + 1) % k
12:     end if
13:     for i ← 0, i < size, i++ do
14:         x ← (begin + i) % k
15:         UPDATE-S-AND-N(x)
16:         UPDATE-VELOCITY(x)
17:     end for
18:     for i ← 0, i < size, i++ do
19:         x ← (begin + i) % k
20:         UPDATE-COORDINATES(x)
21:     end for
22: end procedure
```

The procedures used in the last part are implemented in a similar way as the original algorithm [142]; that is, in line 15, the sampled points are updated, in line 16, the velocities assigned to the MDS points are updated, and in line 20, the coordinates of the MDS points are updated based on these updated velocities. However, the implementation of the UPDATE-VELOCITY procedure (line 16) is slightly different than the original. In particular, in updating the velocities, a decay function is used so that the algorithm forgets "old" forces that might have originated from the elements in $S[]$ and $N[]$ that have been assigned to new query access vectors in the meantime. Note that unless one keeps track of the history

of all the forces that have acted on every point in the system, there is no other way of "undoing" or "forgetting" these "old" forces.

Algorithm 12 Hash Function f

Require:
t: sequence number of a query access vector
Ensure:
$f(t)$ is computed and returned
1: **procedure** F(t)
2: pos $\leftarrow t \,\%\, k$
3: (lo, hi) $\leftarrow$ GROUP-BOUNDS(X[pos])
4: coid $\leftarrow$ CENTROID(lo, hi)
5: **return** HASH(coid) $\%\, b$
6: **end procedure**

Given the sequence number of a query access vector (t), the outcome of the hash function f is determined based on the coordinates of the MDS point that had previously been assigned to the query access vector by the RECONFIGURE procedure. To this end, the coordinate space is divided into b groups containing points with consecutive coordinates such that there are at most $\lceil \frac{k}{b} \rceil$ points in each group. Then, one option is to use the group identifier, which is a number in $\mathbb{Z}_{0 \ldots b-1}$, as the outcome of f, but there is a problem with this naïve implementation. Specifically, it has been observed that even though the *relative* coordinates of MDS points within the "same" group may not change significantly across successive calls to the RECONFIGURE procedure, points within a group, as a whole, may shift. This is an inherent (and in fact, a desirable) property of the incremental algorithm. However, the problem is that there may be far too many cases where the group identifier of a point changes just because the absolute coordinates of the group have changed, even though the point continues to be part of the "same" group. To solve this problem, a method has been developed that computes the centroid within a group by taking the MIN-HASH of the identifiers of points within that group such that these centroids rarely change across successive iterations. Then, the identifier of the centroid, as opposed to its coordinates, is used to compute the group number, hence, the outcome of f. The pseudocode of this procedure is given in Algorithm 12.

One last observation can be made. Internally, MIN-HASH uses multiple hash functions to approximate the degree to which two sets are similar [49]. It is also known that increasing the number of internal hash functions used (within MIN-HASH) should increase the overall

	0	**1**	**2**	**3**	**4**	**5**
$t = 0$	□	□	□	∅		
$t = \lceil k/3 \rceil$		□	□	□	∅	
$t = \lceil 2k/3 \rceil$			□	□	□	∅
$t = k$	∅			□	□	□
$t = \lceil 4k/3 \rceil$	□	∅			□	□
$t = \lceil 5k/3 \rceil$	□	□	∅			□

Figure 4.9: Assuming $b = 3$, □ indicates the allowed locations at each time tick, and ∅ indicates the counter to be reset.

accuracy of the MIN-HASH scheme. However, as unintuitive as it may seem, in TUNABLE-LSH, only a single hash function is used within MIN-HASH, yet, TUNABLE-LSH is still able to achieve sufficiently high accuracy. The reason is as follows. Recall that Algorithm 11 relies on multiple pairwise distances to position every point. Consequently, even though individual pairwise distances may be inaccurate (because only a single hash function is used within MIN-HASH), collectively the errors are cancelled out, and points can be positioned accurately on the MDS coordinate space.

It is easy to show that the RECONFIGURE-F procedure (cf., Algorithm 11) has computational complexity of $O\big(k\, log(\text{heapSize}) + |\vec{q}_t|\big)$, where k denotes the window size, *heapSize* is a constant that determines the maximum size of the max-heap(s) (cf., Table 4.6), which is usually set to $\sqrt{k}$ [142], and $|\vec{q}_t|$ is the number of 1-bits in the query access vector(s). Since the clustering algorithm introduced in this section has $O(\omega)$ computational-complexity, where ω denotes the number of records in the database (i.e., clustering a database of ω records takes $O(\omega)$ time), and since both k and $|\vec{q}_t|$ are independent of, and negligibly small compared to ω, it is safe to assume that RECONFIGURE-F has constant-time complexity (in relation to the clustering algorithm). Consequently, the TUNE procedure (cf. Algorithm 9) is also constant-time.

4.3.5 Resetting Old Entries in Record Utilization Counters in Tunable-LSH

Once the group identifier is computed (cf., Algorithm 12), it should be straightforward to update the record utilization counters (cf., line 8 in Algorithm 9). However, unless the original query access vectors are maintained, there is no way of knowing which counters to decrement when a query access vector becomes stale, as maintaining these original query

access vectors is prohibitively expensive. Therefore, a more efficient scheme has been developed in which old values can also be removed from the record utilization counters.

Instead of maintaining b entries in every record utilization counter, twice as many entries ($2b$) are maintained. Then, whenever the TUNE procedure is called, instead of directly using the outcome of $f(t)$ to locate the counters to be incremented, $f(t)$ is mapped to a location within an "allowed" region of consecutive entries in the record utilization counter (cf., line 8 in Algorithm 9). At every $\lceil \frac{k}{b} \rceil^{\text{th}}$ iteration, this allowed region is shifted by one to the right, wrapping back to the beginning if necessary. Consider Figure 4.9. Assuming that $b = 3$ and that at time $t = 0$ the allowed region spans entries from 0 to $(b - 1)$, at time $t = \lceil \frac{k}{b} \rceil$, the region will span entries from 1 to b; at time $t = k$, the region will span entries from b to $2b-1$; and at time $t = \lceil \frac{4k}{b} \rceil$, the region will span entries 0 and those from $b + 1$ to $2b - 1$.

Since $f(t)$ produces a value between 0 and $b - 1$ (inclusive), whereas the entries are numbered from 0 to $2b - 1$ (inclusive), the RECONFIGURE procedure in Algorithm 9 uses $f(t)$ as follows. If the outcome of $f(t)$ directly corresponds to a location in the allowed region, then it is used. Otherwise, the output is incremented by b (cf., line 8 in Algorithm 9). Whenever the allowed region is shifted to the right, it may land on an already incremented entry. If that is the case, that entry is reset, thereby allowing "old" values forgotten (cf., line 11 in Algorithm 9). These are shown by $\emptyset$ in Figure 4.9. This scheme guarantees any query access pattern that is less than k steps old is remembered, while any query access pattern that is more than $2k$ old is forgotten.

4.3.6 Discussion

In Section (4.3), TUNABLE-LSH has been introduced, which is a tunable locality-sensitive hashing scheme. It has been demonstrated that the original Hamming distances between record utilization vectors can be approximated using TUNABLE-LSH with tight error bounds. An adaptive clustering step has also been introduced, in which queries are pre-clustered based on their access pattern similarity, which not only improves the error bounds of TUNABLE-LSH, but also ensures that these tighter error bounds are maintained even when the query access patterns change. These properties of TUNABLE-LSH enables it to be used for clustering triples in chameleon-db to compute the G-by-Q clusters. TUNABLE-LSH can also be used for determining the serialization order of G-by-Q clusters, however, as indicated earlier, that is beyond the scope of this thesis.

The clustering algorithm that is proposed in this section is oblivious to *how* the last k most representative query access vectors are selected. Therefore, if more effective re-

placement schemes are developed (which is orthogonal to this thesis), they can be easily integrated.

Chapter 5

Query Evaluation and Indexing

This chapter discusses the query evaluation and optimization algorithms used in chameleon-db. Query evaluation in such a system is non-trivial for two reasons:

- While a good algorithm for computing the G-by-Q clusters, such as the ones discussed in Sections 4.2 and 4.3, is necessary to achieve the desired CPU cache and I/O optimizations, it is not sufficient. This is because a query evaluation algorithm that is oblivious to the underlying G-by-Q representation can easily obscure and even reverse the effects of clustering.
- The query plan generator needs to know about the underlying physical layout—in particular, the way the RDF graph is partitioned to come up with a query plan that produces correct results. Even when the underlying G-by-Q clusters change frequently, the system should be able to support the efficient generation and execution of query plans (that produce correct results), and there are trade-offs. On the one hand, it is possible to index and maintain everything about the underlying clusters, in which case, query plans can be generated efficiently, but it becomes difficult to update the clusters. On the other hand, one can choose to create indexes in a minimal fashion, in which case, it is easy to update the clusters, but it becomes difficult to generate query plans.

5.1 Overview of Query Evaluation

Consider two typical ways of evaluating BGPs over fixed, non-adaptive representations: In HolisticEvaluation (Figure 5.1a), the *whole* BGP is evaluated over the *whole* RDF graph [201],

1. Compute the result of the *whole* BGP over the *whole* RDF graph according to Definition 5.

(a) HolisticEvaluation

1. Decompose the BGP into its triple patterns;
2. For each triple pattern,
   (a) Use indexes to locate only those clusters that have a subgraph that matches the triple pattern;
   (b) For each satisfying cluster, compute the result of the triple pattern over that cluster according to Definition 5;
   (c) Take the union of the results; and
3. Join the intermediate results from Step 2.

(b) TriviallyDecomposedEvaluation

1. Use indexes to locate only those clusters that have a subgraph that matches the *whole* BGP;
2. For each satisfying cluster, compute the result of the *whole* BGP over that cluster according to Definition 5; and
3. Take the union of the results.

(c) OptimalEvaluation

1. Preferably, do not decompose the BGP at all; however, if necessary, decompose it into as few number of segments as possible;
2. For each query segment, execute OptimalEvaluation.
3. Join the intermediate results from Step 2.

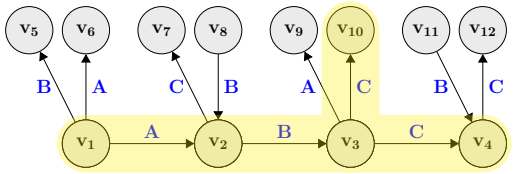
(d) SchemalessEvaluation

Figure 5.1: Alternative query evaluation algorithms

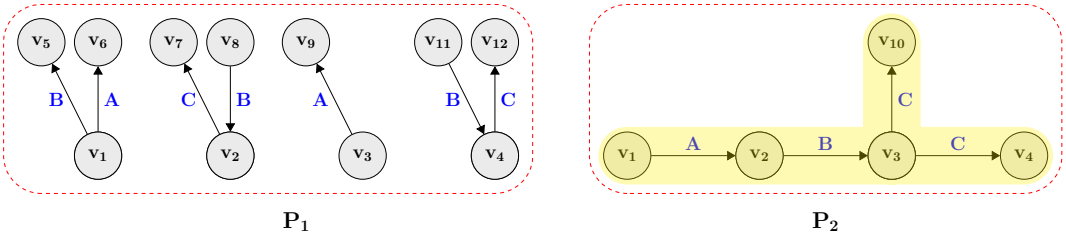
whereas in TriviallyDecomposedEvaluation (Figure 5.1b), the BGP is decomposed into its triple patterns, and each triple pattern is evaluated independently over a clustering of the graph [109].

While both algorithms would produce correct results over the G-by-Q representation, they would be far from the optimal choice. HolisticEvaluation needs to consider the entire RDF graph, which may lead to processing of irrelevant parts. TriviallyDecomposedEvaluation, on the other hand, decomposes the BGP all the way down to its triple patterns, which results in suboptimal performance for reasons discussed in Chapter 1.

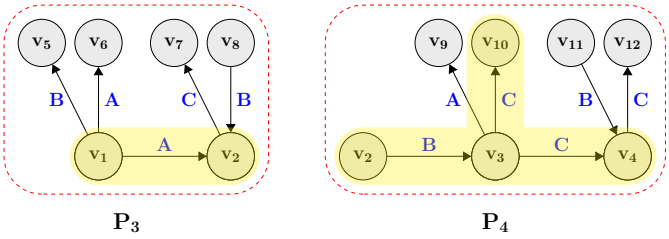
Consider evaluating $Q = ?w \xrightarrow{A} ?x \xrightarrow{B} ?y \xrightarrow{C} ?z$ over Clustering A (cf., Figure 5.2b). Despite the fact that Clustering A is a better choice for Q with respect to the clustering objectives (cf., Section 4.1.2), in Step 2(a) of TriviallyDecomposedEvaluation, indexes cannot efficiently localize query evaluation to only P_2 because P_1 contains at least one edge for each label A,



(a) RDF graph



(b) Clustering A



(c) Clustering B

Figure 5.2: Sample RDF graph and G-by-Q clusterings

B and C. That is, P_1 contains a match for each triple pattern in the query. This results in the generation of irrelevant intermediate result tuples that may remain in the query evaluation pipeline until all the joins in Step 3 are completed. Thus, TriviallyDecomposed-Evaluation not only performs unnecessary computations, but it also results in poor I/O and cache utilization.

In short, neither HolisticEvaluation nor TriviallyDecomposedEvaluation truly exploits the fact that triples in the G-by-Q representation are already being clustered based on the results of the queries in the workload. This is a useful property because given a query from the workload, it is very likely that subgraphs that match the query (i.e., subgraphs that contribute to the result of the query as per Definition 5), are each contained within at most a single (but not necessarily the same) cluster. Recall how the two subgraphs in Clustering A in Figure 5.2b that match Q are each contained in a single cluster.[1] Intuitively, if the aforementioned conditions hold, the correct result of a BGP can be obtained (i) without decomposing the BGP at all, and (ii) by evaluating the whole BGP independently over each cluster in the G-by-Q representation and taking the union of the results, thereby avoiding the join step of TriviallyDecomposedEvaluation. These optimizations are captured in OptimalEvaluation (Figure 5.1c).

OptimalEvaluation is more efficient than both HolisticEvaluation and TriviallyDecomposed-Evaluation. First, when evaluating Q over Clustering A, P_1 can be pruned out already in the first step of the algorithm, which results in good data localization. Second, since the query is evaluated *entirely* over P_2 (as opposed to being decomposed), query evaluation does not produce any irrelevant intermediate result tuples—in fact, there are no intermediate results. Needless to say, by fetching only a single cluster from the storage system, the algorithm also achieves better I/O and cache utilization.

Naturally, the storage advisor strives to compute a G-by-Q clustering such that for every query in the workload, every subgraph that matches that query spans at most one cluster. However, sometimes, this may be too ambitious to achieve. In practice, the above condition may not hold for some queries in the workload, for which OptimalEvaluation would not be applicable. Even in that case, it is argued that reverting all the way down to the decomposition into triple patterns (i.e., TriviallyDecomposedEvaluation) may be unnecessary. Therefore, SchemalessEvaluation is proposed (Figure 5.1d) that encapsulates both TriviallyDecomposedEvaluation and OptimalEvaluation, but depending on the underlying G-by-Q representation, can accommodate a whole range of decompositions in between. Before any formalization, the following questions are answered.

[1] Coincidentally, in this example, both subgraphs are contained also within the *same* cluster, but that is not a necessary condition.

Q1— In SchemalessEvaluation, how can a decomposition of the query be found that produces the *correct* result?

A – A bottom-up approach is followed. That is, the algorithm starts with the decomposition of the query into its triple patterns (i.e., TriviallyDecomposedEvaluation), which always produces correct results regardless of the underlying G-by-Q representation (cf., Theorem 11 in Section 5.2), and rely on equivalence rules to simplify the decomposition. These equivalence rules are conditional, and they exploit various properties about graphs to dynamically determine whether subgraphs that match the query spill into multiple G-by-Q clusters (Section 5.3).

Q2— How can one *efficiently* determine, at runtime, whether any of the matching subgraphs of a query spill into multiple G-by-Q clusters?

A – A lazy approach is followed: in general, there are two cases to consider. Initially (i.e., whenever a query is evaluated for the first time), it is assumed that all of the matching subgraphs of the query spill into multiple clusters. However, as queries are evaluated, summary information is maintained in an index called the *Spill Index*, which is used in firing the conditional equivalence rules in subsequent queries. These equivalence rules are designed such that queries can be optimized efficiently using as little information about the underlying G-by-Q clusters as possible.

Q3— In Step 1 of OptimalEvaluation, how are the relevant clusters determined?

A – Another index, called the *Cluster Index* is used. This index is also constructed in a lazy fashion. That is, given the first query, the index assumes that any of the clusters could be relevant to the evaluation of the query.[2] However, as queries are evaluated, it uncovers more information about the clusters and indexes them.

Q4— What data structures are utilized to facilitate subgraph matching within a cluster?

A – To perform subgraph matching within each cluster, each RDF graph in the G-by-Q cluster is represented using an adjacency list and a variation of Ullmann's algorithm [183] is used. For each vertex $\hat{v}$ in the BGP, the candidate matching vertices in the RDF graph

[2]This is an oversimplification because a signature-index containing information about the labels of edges in each G-by-Q is maintained by default.

are computed. If $\hat{v}$ is a URI or literal, one can directly lookup the vertex in the adjacency list. Otherwise, if $\hat{v}$ is a variable, the algorithm relies on the labels of the edges that are incident on $\hat{v}$ to prune the search space. While it is possible to build an index (other than adjacency lists) over each cluster to facilitate subgraph matching, it is outside the scope of this thesis.

5.2 Building Blocks of Schemaless-Evaluation (SE)

OptimalEvaluation and SchemalessEvaluation rely on two new operations: *prune* (Definition 12) and *clustered-match* (Definition 13). Prune corresponds to Step 1 of OptimalEvaluation, while clustered-match corresponds to Steps 2 and 3.

Definition 12. *Given a clustering $\mathbb{P}$ of an RDF graph and a BGP Q, the* prune *of $\mathbb{P}$ with respect to Q, which is denoted by $\sigma_Q(\mathbb{P})$, is defined as $\sigma_Q(\mathbb{P}) = \{P \in \mathbb{P} \mid [\![Q]\!]_P \neq \emptyset\}$.* □

The key aspect of prune is that unless there is a subgraph in a G-by-Q cluster that matches the *whole* query, that cluster will be discarded, even if the query has partial matches. This property is exploited further when building indexes over the G-by-Q clusters (cf., Section 5.5).

Definition 13. *Let Q be a BGP, $\mathbb{P}$ be a clustering of an RDF graph G and $\mathbb{P}' \subset \mathbb{P}$. The* clustered-match *of Q over $\mathbb{P}'$, denoted as $Q\lfloor\mathbb{P}'\rfloor$, is defined by $[\![Q\lfloor\mathbb{P}'\rfloor]\!]_G = \bigcup_{P \in \mathbb{P}'} [\![Q]\!]_P$.* □

Clustered-match is different from standard match in that $[\![Q]\!]_G = [\![Q\lfloor\mathbb{P}\rfloor]\!]_G$ will hold only if every subgraph of G that matches Q is contained in at most one cluster in $\mathbb{P}$. Note that this is *exactly* the objective of the G-by-Q clustering (cf., Section 4.1). Thus, for most queries in the workload, one might expect to rely on OptimalEvaluation to compute the correct query results. Of course, for cases when the clustering algorithm achieves its objective only partially, the query will have to be decomposed into smaller segments (cf., SchemalessEvaluation) such that for each segment Q_i, $[\![Q_i]\!]_G = [\![Q_i\lfloor\mathbb{P}\rfloor]\!]_G$ holds. To determine a good decomposition (ideally, one with no more than one query segments), the algorithm starts with a decomposition that always produces the correct query result regardless of how the RDF graph is clustered, and computes a better decomposition by dynamically analyzing the current state of the clustering. Next, these concepts are formalized by first defining the so-called SE *expressions*.

SE expressions are defined recursively. Given a BGP Q and a G-by-Q clustering of an RDF graph $\mathbb{P}$, $Q\lfloor\mathbb{P}\rfloor$ and $Q\lfloor\sigma_Q(\mathbb{P})\rfloor$ are SE expressions (they correspond to Steps 1

and 2 of OptimalEvaluation, respectively). If M_1 and M_2 are SE expressions, then so are $(M_1 \cup M_2)$ and $(M_1 \bowtie M_2)$ (they correspond to Step 3 of OptimalEvaluation and Step 3 of SchemalessEvaluation, respectively).

Next, it is shown that with the *trivial decomposition* of a BGP, in which the BGP is decomposed into its triple patterns (Definition 14), the construction of an SE expression M can be guaranteed such that $[\![Q]\!]_G = [\![M]\!]_G$ for any BGP Q and any clustering $\mathbb{P}$ of an RDF graph G (Theorem 11). Henceforth, this expression will be called the *baseline* SE *expression* (Definition 15). Before formally introducing Theorem 11 and proving it, some notation and lemmas are introduced (Lemma 9 and 10).

Definition 14. *Given a BGP $Q = (\hat{V}, \hat{E})$, then the* trivial decomposition *of Q is defined as the set $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ of BGPs, where each $Q_i \in \mathcal{Q}$ contains exactly one edge, the set of edges in each Q_i are disjoint, $\hat{V} = \bigcup_{i=1}^{k} V(Q_i)$, $\hat{E} = \bigcup_{i=1}^{k} E(Q_i)$.* □

Definition 15. *Let Q be a BGP, let G be an RDF graph, and let $\mathbb{P}$ be a clustering of G. If $\{Q_1, \ldots, Q_k\}$ is the trivial decomposition of Q, then the* baseline SE expression *for Q over $\mathbb{P}$ is $Q_1 \lfloor \mathbb{P} \rfloor \bowtie \cdots \bowtie Q_k \lfloor \mathbb{P} \rfloor$.* □

Definition 16. *Given two BGPs Q_A and Q_B, the* concatenation *of Q_A and Q_B, denoted by $Q_A \oplus Q_B$, is defined as a BGP $Q = (\hat{V}, \hat{E})$ such that (i) $\hat{V} = V(Q_A) \cup V(Q_B)$ and (ii) $\hat{E} = E(Q_A) \cup E(Q_B)$.* □

Lemma 9. *Let μ be a solution mapping; and let Q_A and Q_B be two BGPs. Given an RDF graph G, G μ-matches the (concatenated) BGP $Q_A \oplus Q_B$ iff there exist two solution mappings μ_A and μ_B and two RDF graphs G_A and G_B such that $G = G_A \oplus G_B$, G_A μ_A-matches Q_A, G_B μ_B-matches Q_B, $\mu = \mu_A \cup \mu_B$ and $\mu_A \sim \mu_B$.*

Proof 9. *To prove Lemma 9, it needs to be shown that both of the following statements are true:*

- *C1. Given two RDF graphs $G_A = (V_A, E_A)$ and $G_B = (V_B, E_B)$, and two BGPs $Q_A = (\hat{V}_A, \hat{E}_A)$ and $Q_B = (\hat{V}_B, \hat{E}_B)$, if G_A μ_A-matches Q_A and G_B μ_B-matches Q_B, where μ_A and μ_B are two solution mappings such that $\mu_A \sim \mu_B$, then G μ-matches $Q = (\hat{V}, \hat{E})$, where G is an RDF graph with $G = G_A \cup G_B$, $\mu = \mu_A \cup \mu_B$ and $Q = Q_A \oplus Q_B$.*
- *C2. Given an RDF graph $G = (V, E)$, and two BGPs $Q_A = (\hat{V}_A, \hat{E}_A)$ and $Q_B = (\hat{V}_B, \hat{E}_B)$, if G μ-matches $Q_A \oplus Q_B$, where μ is a solution mapping, then there exists two RDF graphs $G_A = (V_A, E_A)$ and $G_B = (V_B, E_B)$, and two solution mappings μ_A and μ_B such that G_A μ_A-matches Q_A, G_B μ_B-matches Q_B, $\mu_A \sim \mu_B$, and $\mu = \mu_A \cup \mu_B$.*

Both of the above statements are proven by construction. To prove that C1 is true, first it is shown that μ satisfies condition (a) in Definition 4 and then two surjective functions M_V and M_E are constructed such that they satisfy condition (b) in Definition 4.

- $\mathrm{dom}(\mu)$ *is the set of variables mentioned in* Q*:*
  - *Since* $\mu = \mu_A \cup \mu_B$*,*

$$\begin{aligned} \mathrm{dom}(\mu) &= \mathrm{dom}(\mu_A \cup \mu_B) \\ &= \mathrm{dom}(\mu_A) \cup \mathrm{dom}(\mu_B) \end{aligned} \tag{5.1}$$

  - *Furthermore,* $Q = Q_A \oplus Q_B$ *implies that the set of variables mentioned in* Q *is the union of the set of variables mentioned in* Q_A *and the set of variables mentioned in* Q_B*.*
  - *Consequently, according to Equation 5.1,* $\mathrm{dom}(\mu)$ *denotes the set of variables mentioned in* Q*.*
- *There are two surjective functions* M_V *and* M_E *such that condition (b) in Definition 4 is satisfied:*
  - *Let* $M_V : (\hat{V}_A \cup \hat{V}_B) \to (V_A \cup V_B)$ *be defined such that*
    - $M_V(\hat{v}) = M_V^A$ *for every* $\hat{v} \in (\hat{V}_A \setminus \hat{V}_B)$*,*
    - $M_V(\hat{v}) = M_V^A$ *for every* $\hat{v} \in (\hat{V}_A \cap \hat{V}_B)$ *and*
    - $M_V(\hat{v}) = M_V^B$ *for every* $\hat{v} \in (\hat{V}_B \setminus \hat{V}_A)$*, where*

    $M_V^A : \hat{V}_A \to V_A$ *and* $M_V^B : \hat{V}_B \to V_B$ *denote the two surjective functions implied by* G_A μ_A*-matches* Q_A *and* G_B μ_B*-matches* Q_B*, respectively.*
  - *Let* $M_E : (\hat{E}_A \cup \hat{E}_B) \to (E_A \cup E_B)$ *be defined such that*
    - $M_E(\hat{e}) = M_E^A$ *for every* $\hat{e} \in (\hat{E}_A \setminus \hat{E}_B)$*,*
    - $M_E(\hat{e}) = M_E^A$ *for every* $\hat{e} \in (\hat{E}_A \cap \hat{E}_B)$ *and*
    - $M_E(\hat{e}) = M_E^B$ *for every* $\hat{e} \in (\hat{E}_B \setminus \hat{E}_A)$*, where*

    $M_E^A : \hat{E}_A \to E_A$ *and* $M_E^B : \hat{E}_B \to E_B$ *denote the two surjective functions implied by* G_A μ_A*-matches* Q_A *and* G_B μ_B*-matches* Q_B*, respectively.*
  - *Note that as long as*
    - $M_V^A(\hat{v}) = M_V^B(\hat{v})$ *for every* $\hat{v} \in (\hat{V}_A \cap \hat{V}_B)$ *and*
    - $M_E^A(\hat{e}) = M_E^B(\hat{e})$ *for every* $\hat{e} \in (\hat{E}_A \cap \hat{E}_B)$*,*

    it can be shown that both M_V *and* M_E *satisfy condition (b) in Definition 4 because of the way* M_V *and* M_E *are defined with respect to* M_V^A*,* M_V^B*,* M_E^A *and* M_E^B*, which already satisfy condition (b) in Definition 4. That is,*

* *It is known that if $\hat{v}$ is a constant, then $M_V^A(\hat{v}) = \hat{v}$ and $M_V^B(\hat{v}) = \hat{v}$, therefore $M_V^A(\hat{v}) = M_V^B(\hat{v})$.*
* *If $\hat{v} \in \mathcal{V}$, then $M_V^A(\hat{v}) = \mu_A(\hat{v})$ and $M_V^B(\hat{v}) = \mu_B(\hat{v})$.*
* *Since $\mu_A \sim \mu_B$, it holds that $\mu_A(?x) = \mu_B(?x)$ for all variables $?x \in \mathrm{dom}(\mu_A) \cap \mathrm{dom}(\mu_B)$, where $\mathrm{dom}(\mu_A) \cap \mathrm{dom}(\mu_B) = (\hat{V}_A \cap \hat{V}_B) \cap \mathcal{V}$.*
* *Consequently, $M_V^A(\hat{v}) = M_V^B(\hat{v})$ for every $\hat{v} \in (\hat{V}_A \cap \hat{V}_B)$ is true.*
* *The proof of $M_E^A(\hat{e}) = M_E^B(\hat{e})$ for every $\hat{e} \in (\hat{E}_A \cap \hat{E}_B)$ follows the same logic.*

For C2, only a proof sketch is included.

- *It is possible to construct a surjective function $M_E^A : \hat{E}_A \to E_A$ by projecting M_E onto the domain $\hat{E}_A$.*
- *Likewise, a surjective function $M_V^A : \hat{V}_A \to V_A$ can be constructed by projecting M_V onto the domain $\hat{V}_A$.*
- *It is not difficult to see that there exists a solution mapping μ_A, such that $\mu_A \sim \mu$ and $\mathrm{dom}(\mu_A)$ corresponds to the set of variables mentioned in Q_A, for which condition (b) in Definition 4 is satisfied.*
  - *Perhaps the more difficult part is to prove that for each $(\hat{e}_1, e_2) \in \hat{E}_A \times E_A$ with $M_E^A(\hat{e}_1) = e_2$: $\hat{e}_1 = (\hat{s}_1, \hat{p}_1, \hat{o}_1)$ and $e_2 = (s_2, p_2, o_2)$, if $M_V^A(\hat{s}_1) = s_2$, then $M_V^A(\hat{o}_1) = o_2$.*
  - *However, since $\hat{V}_A$ and $\hat{E}_A$ define a BGP (i.e., Q_A), it can easily be shown that both the source and the target of each edge $\hat{e} \in \hat{E}_A$ have to be elements of $\hat{V}_A$, which makes the proof possible.*
  - *The observation above can also be utilized in order to show that $G_A = (V_A, E_A)$ is an RDF graph.*
- *Similar arguments can be made about the existence of an RDF graph G_B and a solution mapping μ_B such that G_B μ_B-matches Q_B.*
- *Since (i) $\mu_A \sim \mu$ and $\mu_B \sim \mu$, and (ii) $\mathrm{dom}(\mu_A) \subseteq \mathrm{dom}(\mu)$ and $\mathrm{dom}(\mu_B) \subseteq \mathrm{dom}(\mu)$, it also holds that $\mu_A \sim \mu_B$.*
- *Furthermore, $Q = Q_A \oplus Q_B$ implies that $\mathrm{dom}(\mu) = \mathrm{dom}(\mu_A) \cup \mathrm{dom}(\mu_B)$, hence, it also holds that $\mu = \mu_A \cup \mu_B$.*
- *Consequently, Statement C2 is true.* □

Lemma 10. *Given an RDF graph G and two BGPs Q_A and Q_B, $[\![Q_A \oplus Q_B]\!]_G = [\![Q_A]\!]_G \bowtie [\![Q_B]\!]_G$.*

Proof 10. *To prove Lemma 10, it needs to be shown that both of the following statements hold:*

L1. If μ is a solution mapping such that $\mu \in [\![Q_A \oplus Q_B]\!]_G$, then $\mu \in \big([\![Q_A]\!]_G \bowtie [\![Q_B]\!]_G\big)$; and
L2. If μ is a solution mapping such that $\mu \in \big([\![Q_A]\!]_G \bowtie [\![Q_B]\!]_G\big)$, then $\mu \in [\![Q_A \oplus Q_B]\!]_G$.

Both of these statements are proven by contradiction.

To prove L1:

- *Assume there exists a solution mapping μ such that $\mu \in [\![Q_A \oplus Q_B]\!]_G$, but $\mu \notin \big([\![Q_A]\!]_G \bowtie [\![Q_B]\!]_G\big)$.*
- *By Definition 5, there exists an RDF graph G' that is a subgraph of G such that G' μ-matches $Q_A \oplus Q_B$ where $\mu \notin [\![Q_A]\!]_G \bowtie [\![Q_B]\!]_G$.*
- *Then, according to Lemma 9, there exists two RDF graphs G_A and G_B that are subgraphs of G' such that G_A μ_A-matches Q_A and G_B μ_B-matches Q_B, where μ_A and μ_B are two solution mappings such that $\mu = \mu_A \cup \mu_B$ and $\mu_A \sim \mu_B$.*
- *Since G_A and G_B are also subgraphs of G (transitively), according to Definition 5, $\mu_A \in [\![Q_A]\!]_G$ and $\mu_B \in [\![Q_B]\!]_G$.*
- *However, since $\mu = \mu_A \cup \mu_B$ and $\mu_A \sim \mu_B$, according to the standard definition of join ($\bowtie$) in SPARQL algebra [157], μ must also be an element of $[\![Q_A]\!]_G \bowtie [\![Q_B]\!]_G$, which is a contradiction.*
- *Consequently, Statement L1 must hold.*

To prove L2:

- *Assume there exists a solution mapping μ such that $\mu \in \big([\![Q_A]\!]_G \bowtie [\![Q_B]\!]_G\big)$, but $\mu \notin [\![Q_A \oplus Q_B]\!]_G$.*
- *Based on the standard definition of join ($\bowtie$) in SPARQL algebra [157], there must exist two solution mappings $\mu_A \in [\![Q_A]\!]_G$ and $\mu_B \in [\![Q_B]\!]_G$ such that $\mu = \mu_A \cup \mu_B$ and $\mu_A \sim \mu_B$.*
- *According to Definition 5, there must exist two RDF graphs G_A and G_B such that (i) G_A and G_B are both subgraphs of G, (ii) G_A μ_A-matches Q_A, and (iii) G_B μ_B-matches Q_B.*
- *Since $\mu = \mu_A \cup \mu_B$ and $\mu_A \sim \mu_B$, Lemma 9 suggests that there also exists an RDF graph G' such that $G' = G_A \cup G_B$ and G' μ-matches $Q_A \oplus Q_B$.*
- *Since both G_A and G_B are subgraphs of G, so is G', and according to Definition 5, $\mu \in [\![Q_A \oplus Q_B]\!]_G$, however, this is a contradiction.*

- *Consequently, Statement L2 must hold.* □

Now, Theorem 11 can be introduced and proven.

Theorem 11. *Given a BGP Q, an RDF graph G and a clustering $\mathbb{P}$ of G, $[\![Q]\!]_G = [\![M]\!]_G$, where M is the baseline* SE *expression for Q over $\mathbb{P}$.*

Proof 11. *It needs to be shown that given a BGP Q, an RDF graph G and a clustering $\mathbb{P}$ of G, $[\![Q]\!]_G = M$ holds, where M is the baseline* SE *expression for Q over $\mathbb{P}$. Let $\mathrm{TD}(Q) = \{Q_1, \ldots, Q_k\}$ be the trivial decomposition of Q, and let $\mathbb{P} = \{P_1, \ldots, P_m\}$. Note that according to Definition 13, for each $Q_i \in \mathrm{TD}(Q)$,*

$$Q_i\lfloor\mathbb{P}\rfloor = [\![Q_i]\!]_{P_1} \cup \cdots \cup [\![Q_i]\!]_{P_m}. \tag{5.2}$$

and based on Definition 14, it is also known that each $Q_i \in \mathrm{TD}(Q)$ contains exactly one edge, therefore

$$[\![Q_i]\!]_{P_1} \cup \cdots \cup [\![Q_i]\!]_{P_m} = [\![Q_i]\!]_{P_1 \cup \cdots \cup P_m}$$

and

$$Q_i\lfloor\mathbb{P}\rfloor = [\![Q_i]\!]_{P_1 \cup \cdots \cup P_m}. \tag{5.3}$$

Since $\mathbb{P} = \{P_1, \ldots, P_m\}$ is a clustering of G, by Definition 6, $P_1 \cup \cdots \cup P_m = G$, hence,

$$Q_i\lfloor\mathbb{P}\rfloor = [\![Q_i]\!]_G. \tag{5.4}$$

Substituting values in the baseline SE *expression with the right hand side of Equation 5.4, one gets*

$$M = [\![Q_1]\!]_G \bowtie \cdots \bowtie [\![Q_k]\!]_G. \tag{5.5}$$

By recursively applying Lemma 10, one gets

$$M = [\![Q_1 \oplus \cdots \oplus Q_k]\!]_G. \tag{5.6}$$

Since $Q = Q_1 \oplus \cdots \oplus Q_k$ by definition of trivial decomposition, $[\![Q]\!]_G = M$ holds. □

	Name	Equivalence Rules	Condition
1	Expansion	$[\![Q_A\lfloor\mathbb{P}\rfloor]\!]_G = \bigcup_{i=1}^{m} [\![Q_A\lfloor\mathbb{P}_i\rfloor]\!]_G$	$\bigcup_{i=1}^{m} \mathbb{P}_i = \mathbb{P}$
2	Join elimination*	$[\![Q_A\lfloor\mathbb{P}_1\rfloor \bowtie Q_B\lfloor\mathbb{P}_2\rfloor]\!]_G = \emptyset$	Thm. 12
3	Join reduction*	$[\![Q_A\lfloor\mathbb{P}_1\rfloor \bowtie Q_B\lfloor\mathbb{P}_1\rfloor]\!]_G$ $= [\![(Q_A \oplus Q_B)\lfloor\mathbb{P}_1\rfloor]\!]_G$	Thm. 13
4	Identity ($\bowtie$)	$\Omega_1 \bowtie \emptyset = \emptyset \bowtie \Omega_1 = \emptyset$	$\Omega_1, \Omega_2, \Omega_3$
5	Identity ($\cup$)	$\Omega_1 \cup \emptyset = \emptyset \cup \Omega_1 = \Omega_1$	are sets of
6	Associativity ($\bowtie$)	$\Omega_1 \bowtie (\Omega_2 \bowtie \Omega_3)$ $= (\Omega_1 \bowtie \Omega_2) \bowtie \Omega_3$	solution mappings
7	Associativity ($\cup$)	$\Omega_1 \cup (\Omega_2 \cup \Omega_3)$ $= (\Omega_1 \cup \Omega_2) \cup \Omega_3$	
8	Distributivity ($\bowtie$ over $\cup$)	$\Omega_1 \bowtie (\Omega_2 \cup \Omega_3)$ $= (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \bowtie \Omega_3)$	
9	Reflexivity	$\Omega_1 \bowtie \Omega_2 = \Omega_2 \bowtie \Omega_1$	

Table 5.1: Equivalence rules that are applicable to the evaluation of SE expressions ($\mathbb{P}_1, \ldots, \mathbb{P}_m$ represent sets of RDF graphs).

5.3 Query Rewriting Rules

To realize the aforementioned rewrite of the baseline expression into an equivalent expression with fewer number of join operations, equivalence rules are introduced that are in two categories: generic and conditional. Generic rules are applicable irrespective of how the RDF graph is clustered, whereas the applicability of a conditional rule depends on whether the clustering satisfies certain conditions.

Assuming Q_A and Q_B are two BGPs, let $\mathbb{P}$ be a clustering of an RDF graph with subsets $\mathbb{P}_1, \ldots, \mathbb{P}_m$ ($\mathbb{P}_i \subseteq \mathbb{P}$ for all $i \in \{1, \ldots, m\}$). Table 5.1 lists the equivalence rules. Rules 1–3 are specific to the clustered-match operation, whereas rules 4–9 are derived from SPARQL algebra [29]. Rules that are marked with an asterisk (*) are conditional. Observe that the expansion rule relies on a condition that is independent of the way the graph is clustered. In other words, for any clustering $\mathbb{P}$ of an RDF graph, one may generate some $\mathbb{P}_1, \ldots, \mathbb{P}_m$, such that the condition is satisfied. On the contrary, the conditions in join elimination and join reduction are directly related to the way the graph is clustered; therefore, they need to be checked every time a query is evaluated. The following two theorems formalize these conditions and show their correctness.

Theorem 12. *Given a clustering $\mathbb{P}$ of an RDF graph G and two BGPs Q_A and Q_B with $V(Q_A) \cap V(Q_B) \neq \emptyset$, let $I = \bigcup_{(P_i,P_j)\in\mathbb{P}_1\times\mathbb{P}_2} V(P_i) \cap V(P_j)$, where $\mathbb{P}_1, \mathbb{P}_2 \subseteq \mathbb{P}$. Then, $[\![Q_A\lfloor\mathbb{P}_1\rfloor \bowtie Q_B\lfloor\mathbb{P}_2\rfloor]\!]_G = \emptyset$ if, for each vertex $v \in I$, there exists a vertex $\hat{v} \in V(Q_A) \cap V(Q_B)$*

and an edge $\hat{e} \in inc(Q_A, \hat{v}) \cup inc(Q_B, \hat{v})$ *such that* $\hat{e}$ *is not compatible (cf., Definition 3) with any edge in* $\bigcup_{P \in \mathbb{P}_1 \cup \mathbb{P}_2} inc(P, v)$*, where* $inc(G, v)$ *denotes the set of edges that are incident on a vertex* v*.*

Proof 12. *It needs to be shown that* $Q_A \lfloor \mathbb{P}_1 \rfloor \bowtie Q_B \lfloor \mathbb{P}_2 \rfloor = \emptyset$ *holds. Note that according to Definition 13, a semantically equivalent expression that one can prove is* $\bigcup_{(P_i, P_j) \in \mathbb{P}_1 \times \mathbb{P}_2} [\![Q_A]\!]_{P_i} \bowtie [\![Q_B]\!]_{P_j} = \emptyset$*. In the next part, the latter statement is proven by contradiction.*

S1. Assume that there exist two clusters $P_i \in \mathbb{P}_1$ *and* $P_j \in \mathbb{P}_2$ *such that* $[\![Q_A]\!]_{P_i} \bowtie [\![Q_B]\!]_{P_j} \neq \emptyset$*.*

S2. Based on the standard SPARQL semantics [157], since the join in the above expression returns a non-empty set of solution mappings, there must exist a solution mapping $\mu \in [\![Q_A]\!]_{P_i} \bowtie [\![Q_B]\!]_{P_j}$ *and two solution mappings* $\mu_A \in [\![Q_A]\!]_{P_i}$ *and* $\mu_B \in [\![Q_B]\!]_{P_j}$ *such that* $\mu = \mu_A \cup \mu_B$ *and* $\mu_A \sim \mu_B$*.*

S3. Then, according to Definition 5, there must exist two RDF graphs G_A *and* G_B *such that (i)* G_A *is a subgraph of* P_i *and* G_A μ_A*-matches* Q_A*, and (ii)* G_B *is a subgraph of* P_j *and* G_B μ_B*-matches* Q_B*.*

S4. Lemma 9 suggests that there also exists an RDF graph G' *such that* $G' = G_A \cup G_B$ *and* G' μ*-matches* $Q_A \oplus Q_B$*.*

S5. Statements (C) and (D), together with Definitions 2 and 4 imply that for every vertex $\hat{v} \in V(Q_A) \cap V(Q_B)$

- $inc(Q_A, \hat{v}) \neq \emptyset$ *and* $inc(Q_B, \hat{v}) \neq \emptyset$ *(because according to Definition 2, a vertex in a BGP cannot exist without an edge incident on it); and*
- *there exists a vertex* $v \in V(G_A) \cap V(G_B)$ *to which* $\hat{v}$ *is mapped (because the three surjective functions* $M_V^{G_A}$*,* $M_V^{G_B}$ *and* $M_V^{G'}$ *implied by* G_A μ_A*-matches* Q_A*,* G_B μ_B*-matches* Q_B *and* G' μ*-matches* $Q_A \oplus Q_B$ *must agree on all common vertices) such that*
  - *every edge* $\hat{e} \in inc(Q_A, \hat{v})$ *is compatible with an edge from* $e \in inc(G_A, v)$*;*
  - *likewise, every edge* $\hat{e} \in inc(Q_B, \hat{v})$ *is compatible with an edge from* $e \in inc(G_B, v)$ *(which follows the previous statement and Definition 4); and*

S6. In other words, for every vertex $\hat{v} \in V(Q_A) \cap V(Q_B)$*, there exists a vertex* $v \in V(G_A) \cap V(G_B)$ *such that every edge* $\hat{e} \in inc(Q_A, \hat{v}) \cup inc(Q_B, \hat{v})$ *is compatible with an edge from* $inc(G_A, v) \cup inc(G_B, v)$*.*

S7. Recall that $V(Q_A) \cap V(Q_B) \neq \emptyset$*, therefore, Statement (F) also implies that there exists a vertex* $v \in V(G_A) \cap V(G_B)$ *and a vertex* $\hat{v} \in V(Q_A) \cap V(Q_B)$ *such that every edge* $\hat{e} \in inc(Q_A, \hat{v}) \cup inc(Q_B, \hat{v})$ *is compatible with an edge from* $inc(G_A, v) \cup inc(G_B, v)$*.*

S8. However, Statement (G) contradicts the conditions in Theorem 12, therefore proof-by-contradiction suggests Statement (A) does not hold under these conditions.

S9. Consequently, $\bigcup_{(P_i,P_j)\in\mathbb{P}_1\times\mathbb{P}_2}[\![Q_A]\!]_{P_i} \bowtie [\![Q_B]\!]_{P_j} = \emptyset$ *and Theorem 12 holds.* □

Theorem 13. *Given a clustering* $\mathbb{P}$ *of an RDF graph* G *and two BGPs* Q_A *and* Q_B *with* $V(Q_A) \cap V(Q_B) \neq \emptyset$, $[\![Q_A\lfloor\mathbb{P}\rfloor \bowtie Q_B\lfloor\mathbb{P}\rfloor]\!]_G = [\![(Q_A \oplus Q_B)\lfloor\mathbb{P}\rfloor]\!]_G$ *if, for each vertex* v *such that* $\left|cont(\mathbb{P}, v)\right| > 1$ *(where* $cont(\mathbb{P}, v)$ *denotes the subset of clusters in* $\mathbb{P}$ *that contain* v*), either*

(i) there exists a vertex $\hat{v} \in V(Q_A) \cap V(Q_B)$ *and an edge* $\hat{e} \in inc(Q_A, \hat{v}) \cup inc(Q_B, \hat{v})$ *such that* $\hat{e}$ *is not compatible with any edge from* $\bigcup_{P\in\mathbb{P}} inc(P, v)$*, or*

(ii) there exists a single cluster $\bar{P} \in \mathbb{P}$ *such that for every edge* $e \in \bigcup_{P\in\mathbb{P}} inc(P, v)$ *and for every vertex* $\hat{v} \in V(Q_A) \cap V(Q_B)$*, if* e *is compatible with an edge from* $inc(Q_A, \hat{v}) \cup inc(Q_B, \hat{v})$*, then* $cont(\mathbb{P}, e) = \{\bar{P}\}$.

Proof 13. *It needs to be shown that*

$$Q_A\lfloor\mathbb{P}\rfloor \bowtie Q_B\lfloor\mathbb{P}\rfloor = Q_A \oplus Q_B\lfloor\mathbb{P}\rfloor \tag{5.7}$$

holds. According to Definition 13, Equation 5.7 is equivalent to

$$\bigcup_{(P_i,P_j)\in(\mathbb{P}\times\mathbb{P})} [\![Q_A]\!]_{P_i} \bowtie [\![Q_B]\!]_{P_j} = \bigcup_{P\in\mathbb{P}} [\![Q_A \oplus Q_B]\!]_P. \tag{5.8}$$

Furthermore, according to Lemma 10, the right hand side of Equation 5.8 can be rewritten as follows:

$$\bigcup_{(P_i,P_j)\in(\mathbb{P}\times\mathbb{P})} [\![Q_A]\!]_{P_i} \bowtie [\![Q_B]\!]_{P_j} = \bigcup_{P\in\mathbb{P}} [\![Q_A]\!]_P \bowtie [\![Q_B]\!]_P, \tag{5.9}$$

Given the equivalence of Equation 5.7 and Equation 5.9, Theorem 13 is proven by showing that Equation 5.9 holds. Note that Equation 5.9 holds iff

$$[\![Q_A]\!]_{P_i} \bowtie [\![Q_B]\!]_{P_j} = \emptyset \tag{5.10}$$

for all $(P_i, P_j) \in \mathbb{P} \times \mathbb{P}$ *with* $P_i \neq P_j$*. Therefore, it needs to be shown that Equation 5.10 holds both for sub-condition (i) and sub-condition (ii) in Theorem 13. The proof of sub-condition (i) follows the same steps as the proof of Theorem 12, therefore, it is omitted. Sub-condition (ii) is proven using proof-by-contradiction as follows:*

$inc(\mathbb{P}, v_1)$	$inc(\mathbb{P}, v_2)$	$inc(\mathbb{P}, v_3)$	in	$inc(\mathbb{P}, v_4)$
$v_1 \xrightarrow{A} v_2$	$v_2 \xleftarrow{A} v_1$	$v_3 \xrightarrow{A} v_9$	P_1	$v_4 \xrightarrow{B} v_{11}$
$v_1 \xrightarrow{A} v_6$	$v_2 \xrightarrow{B} v_3$	$v_3 \xleftarrow{B} v_2$	P_2	$v_4 \xleftarrow{C} v_3$
$v_1 \xrightarrow{B} v_5$	$v_2 \xrightarrow{B} v_8$	$v_3 \xrightarrow{C} v_4$	P_2	$v_4 \xrightarrow{C}$ "10"
	$v_2 \xrightarrow{C} v_7$	$v_3 \xrightarrow{C} v_{10}$	P_2	

Table 5.2: Incident edges on v_1–v_4 in Clustering A (Figure 5.2b)

S1. Assume that there exist two clusters $P_i, P_j \in \mathbb{P}$, where $P_i \neq P_j$, such that $[\![Q_A]\!]_{P_i} \bowtie [\![Q_B]\!]_{P_j} \neq \emptyset$.
S2. Same as Statements (B)–(E) in the proof of Theorem 12.
S3. Furthermore, according to Definition 4, for every $\hat{v} \in V(Q_A) \cap V(Q_B)$, there exists an edge e_A in $inc(G_A, v)$, which is compatible with an edge in $inc(Q_A, \hat{v})$, such that $cont(\mathbb{P}, e_A) = P_i$ and an edge e_B in $inc(G_B, v)$, which is compatible with an edge in $inc(Q_B, \hat{v})$, such that $cont(\mathbb{P}, e_B) = P_j$.
S4. However, $P_i \neq P_j$, which contradicts condition (ii) in Theorem 13.
S5. Proof-by-contradiction suggests Statement (A) does not hold under any of the above conditions.
S6. Consequently, $\bigcup_{(P_i,P_j)\in\mathbb{P}_1\times\mathbb{P}_2} [\![Q_A]\!]_{P_i} \bowtie [\![Q_B]\!]_{P_j} = \emptyset$ and Theorem 13 holds. □

Let Q be $Q_A \oplus Q_B$ (i.e., the concatenation of two BGPs). Theorem 13 formalizes that if Q does not have any matching subgraph that spans multiple clusters, then Q can be evaluated by (i) computing the matching subgraphs of Q within each cluster in isolation, and (ii) taking the union of the results from step (i), thereby omitting the join. Thus, if partial matches of Q_A and Q_B do not join across clusters, query evaluation can be simplified. Condition (i) guarantees that clusters in consideration do not share common vertices (or if there are such vertices then they are not related to the evaluation of Q); and condition (ii) says that if there is such a vertex, the edges incident on that vertex do not match the query edges. Under these circumstances, Q cannot have any matching subgraph that spans multiple clusters and the join can be eliminated.

Next, the earlier query evaluation example is revisited to demonstrate how join reduction can be applied to the schemaless-evaluation of query $?w \xrightarrow{A} ?x \xrightarrow{B} ?y \xrightarrow{C} ?z$ over the clustering in Figure 5.2b. The baseline SE expression for this BGP is $Q_1\lfloor\mathbb{P}\rfloor \bowtie Q_2\lfloor\mathbb{P}\rfloor \bowtie Q_3\lfloor\mathbb{P}\rfloor$, where $?w \xrightarrow{A} ?x$, $?x \xrightarrow{B} ?y$, $?y \xrightarrow{C} ?z$ are the three BGPs Q_1, Q_2 and Q_3 in the trivial decomposition of the query, and $\mathbb{P}$ consists of P_1 and P_2 in Figure 5.2b. For simplicity, prune operations are ignored for now. If $[\![Q_2\lfloor\mathbb{P}\rfloor \bowtie Q_3\lfloor\mathbb{P}\rfloor]\!]_G = [\![(Q_2 \oplus Q_3)\lfloor\mathbb{P}\rfloor]\!]_G$, this baseline SE expression can be rewritten as $Q_1\lfloor\mathbb{P}\rfloor \bowtie (Q_2 \oplus Q_3)\lfloor\mathbb{P}\rfloor$. For the given clustering, the

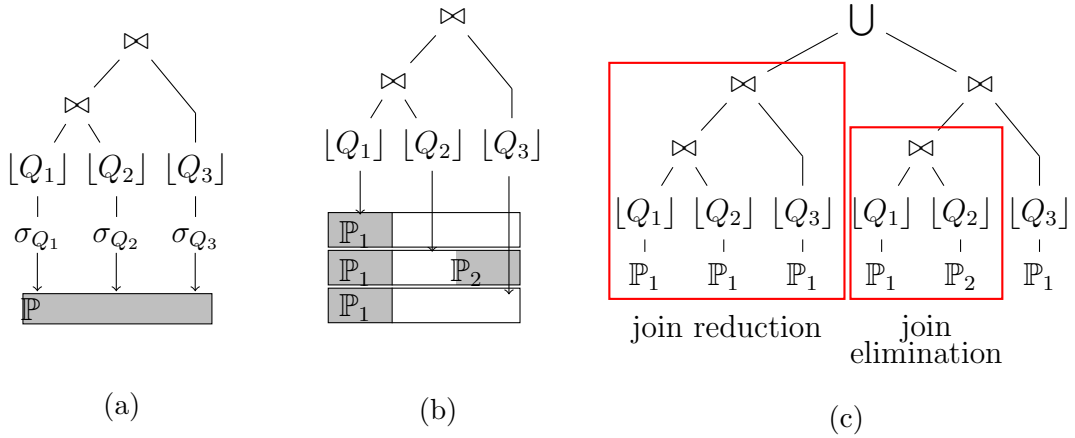


Figure 5.3: Illustration of query rewriting and optimization.

four vertices v_1, v_2, v_3 and v_4 exist in multiple clusters. Therefore, the conditions in Theorem 13 need to be checked. Note that $inc(Q_2, ?y) \cup inc(Q_3, ?y)$ consists of two edges, namely, $(?x, B, ?y)$ and $(?y, C, ?z)$. Condition (i) holds for v_1 because $(?y, C, ?z)$ is not compatible with any of the edges in $inc(\mathbb{P}, v_1)$, which is illustrated in Table 5.2. For v_2, $(?y, B, ?z)$ is not compatible with any of the edges in $inc(\mathbb{P}, v_2)$ due to the direction of edges. The same argument applies to v_4. Regarding v_3, both $(?x, B, ?y)$ and $(?y, C, ?z)$ have at least one compatible edge; therefore, condition (ii) also needs to be checked for v_3. Since all compatible edges are from the same cluster, namely, P_2, the baseline SE expression can be simplified to $Q_1\lfloor\mathbb{P}\rfloor \bowtie (Q_2 \oplus Q_3)\lfloor\mathbb{P}\rfloor$. Continuing with the process, the expression can be further simplified to $(Q_1 \oplus Q_2 \oplus Q_3)\lfloor\mathbb{P}\rfloor$.

5.4 Query Rewriting Algorithm

The algorithm for rewriting a baseline SE expression proceeds in three phases (cf., Algorithm 13 and 14). This algorithm is described using the example illustrated in Figure 5.3. Consider a baseline SE expression: $Q_1\lfloor\sigma_{Q_1}(\mathbb{P})\rfloor \bowtie Q_2\lfloor\sigma_{Q_2}(\mathbb{P})\rfloor \bowtie Q_3\lfloor\sigma_{Q_3}(\mathbb{P})\rfloor$ (Figure 5.3a). First, the joins in the baseline expression are reordered according to their estimated selectivities [176]. Second, by using generic equivalence rules, the expression is transformed into a *canonical form*. An SE expression is in canonical form if it consists of the union of a set of sub-expressions $T_1 \cup \cdots \cup T_m$, where each sub-expression is made up of the exact same set of clustered-match operations, which differ only in the clusters they operate on.

To compute the canonical SE expression, first, each prune operation is evaluated, producing multiple sets of clusters with one set for each prune operation (Figure 5.3b). As illustrated in Figure 5.3b, these sets of clusters are factorized into maximal common subsets such that factorization produces as few segments as possible. For example, let $\mathbb{P} = \{P_a, P_b, P_c, P_d\}$ denote the set of clusters in the example; and assume that $\{P_a, P_b\}$ is a prune of $\mathbb{P}$ with respect to Q_1 and $\{P_a, P_b, P_d\}$ and $\{P_a, P_b\}$ are prunes with respect to Q_2 and Q_3, respectively. Then, $\mathbb{P}_1 = \{P_a, P_b\}$ and $\mathbb{P}_2 = \{P_d\}$ are the maximal common subsets. In the next step, each clustered-match operation is expanded across the corresponding subsets of clusters using Rule 1. Rules 4–9 are applied to the nodes of the expression-tree in a bottom-up fashion, which is repeated until no further rewriting is possible. At this stage, the canonical expression is produced (Figure 5.3c).

In the third phase, each sub-expression in the canonical form is optimized independently using conditional rules (i.e., Rules 2–3) as well as Rules 4–9. In this regard, join reduction and join elimination are applied recursively to the nodes of each sub-expression until the original query is decomposed into as few segments as possible. The right-hand side of union is eliminated using join elimination (Figure 5.3c) and the remaining expression is simplified to $(Q_1 \oplus Q_2 \oplus Q_3)\lfloor S_1 \rfloor$ using join reduction.

In summary, query evaluation corresponds to query plan generation and optimization followed by plan execution. Consequently, to evaluate a BGP, for that given BGP, an SE expression is generated (which of the equivalent expressions the system chooses is the topic of Section 5.3). Figure 5.3a illustrates a tree representation of such an SE expression. Then, each sub-expression of the form $\sigma_{Q_i}(\mathbb{P})$ is evaluated by pruning out the irrelevant clusters using the Cluster Index. Consequently, each sub-expression of the form $Q_i\lfloor\sigma_{Q_i}(\mathbb{P})\rfloor$ is simplified to $Q_i\lfloor\mathbb{P}_i\rfloor$, where $\mathbb{P}_i \subseteq \mathbb{P}$. Then, for each resulting sub-expression $Q_i\lfloor\mathbb{P}_i\rfloor$, (i) the sub-expression is evaluated in isolation on each cluster using a standard subgraph matching technique [183], and (ii) the union of the results from each evaluation is computed. In the subsequent steps, intermediate tuples from the evaluation of each sub-expression are joined or unioned according to the standard definitions in SPARQL algebra [157].

The computational complexity of Get-Baseline-Expression (cf., Algorithm 13) is $O\big(|Q|\,log(|Q|) + |Q|\,|\text{cIndex}|\big)$, where $|Q|$ denotes the number of triple patterns in Q and $|\text{cIndex}|$ denotes the number of nodes in the Cluster Index. $O\big(|Q|\,log(|Q|)\big)$ comes from line 13 and $O\big(|Q|\,|\text{cIndex}|\big)$ comes from lines 25–28 of Algorithm 13.

Algorithm 13 Get-Baseline-Expression

Require:
Q: Basic graph pattern Q for which query plan is to be generated.
cIndex: Pointer to the Cluster Index object.
Ensure:
The query plan tree for the baseline SE expression for Q is generated and returned.

```
1:  procedure GET-BASELINE-EXPRESSION(Q, cIndex)
2:      ▷ Initialize variables
3:      int i ← 0
4:      vector⟨float, tPattern⟩ tpSelArray ← ∅

5:      ▷ Decompose Q into its triple patterns, and
6:      ▷ ... order the triple patterns based on their estimated selectivities.
7:      vector⟨tPattern⟩ tpArray ← GET-TRIPLE-PATTERNS(Q)
8:      for i < tpArray.length do
9:          float sel ← ESTIMATE-SEL(tpArray[i])
10:         tpSelArray[i] ← ⟨sel, tpArray[i]⟩
11:         i++
12:     end for
13:     SORT(tpSelArray, ASCENDING)

14:     node root ← NULL
15:     vector⟨node⟩ unionRoots ← ∅

16:     ▷ Generate query plan tree for the baseline SE expression.
17:     i ← 0
18:     for i < tpSelArray.length do
19:         int j ← 0
20:         vector⟨node⟩ nodeArray ← ∅

21:         ▷ For each segment of G-by-Q clusters that have a match for Q,
22:         ▷ ... generate a plan node for clustered-match operation (cf., Definition 13).
23:         tPattern tp ← tpSelArray[i].second
24:         vector⟨int⟩ segArray ← clusterIndex.GET-SEGMENTS(tp)
25:         for j < segArray.length do
26:             nodeArray[j] ← MAKE-MATCH-NODE(tp, segArray[j])
27:             j++
28:         end for
```

```
29:         ▷ Combine clustered-match operations using union.
30:         if nodeArray.length > 0 then
31:             unionRoots[i] ← nodeArray[0]
32:             j ← 1
33:             for j < nodeArray.length do
34:                 unionRoots[i] ← Make-Union-Node(unionRoots[i], nodeArray[j])
35:                 j++
36:             end for
37:         else
38:             unionRoots[i] ← Make-Empty-Node()
39:         end if

40:         ▷ Combine the root union expressions using join.
41:         if i == 0 then
42:             root ← unionRoots[i]
43:         else
44:             root ← Make-Join-Node(root, unionRoots[i])
45:         end if

46:         i++
47:     end for

48:     return root
49: end procedure
```

Algorithm 14 Rewrite-Expression

Require:

root: Denotes the root of the query plan tree storing the baseline SE expression.

ruleList: List of rules that need to be applied to the baseline SE expression. Each rule object denotes one of the equivalence rules in Table 5.1.

Ensure:

The baseline SE expression is rewritten according to the given equivalence rules. The root of the new query plan tree is returned.

```
1:  procedure REWRITE-EXPRESSION(root, ruleList)
2:      bool updated ← false
3:      ▷ Repeat until query plan cannot be updated
4:      do
5:          updated ← false
6:          stack⟨node⟩ st ← ∅
7:          st.PUSH(root)
8:          while !st.EMPTY() do
9:              node n ← st.POP()
10:             bool ruleFound ← false
11:             do
12:                 for all rule ∈ ruleList do
13:                     ▷ Check if rule is applicable to query plan sub-tree rooted at n
14:                     if ISAPPLICABLE(rule, n) then
15:                         ▷ Modify sub-tree based on equivalence rule
16:                         n ← APPLYRULE(rule, n)
17:                         ▷ If n is the root of the entire query plan tree
18:                         ▷ . . . make sure to update the root variable
19:                         if n.parent == NULL then
20:                             root ← n
21:                         end if
22:                         ruleFound ← true
23:                         updated ← true
24:                         break
25:                     end if
26:                 end for
27:             while ruleFound
```

```
28:             ▷ Push children of n to stack for subsequent iterations
29:             if n.rightChild != NULL then
30:                 st.PUSH(n.rightChild)
31:             end if
32:             if n.leftChild != NULL then
33:                 st.PUSH(n.leftChild)
34:             end if
35:         end while
36:     while updated
37:     return root
38: end procedure
```

5.5 Partial, Adaptive Indexing

This section describes the indexing approach in chameleon-db. chameleon-db uses two types of indexes: the *Spill Index* is used during query plan generation and the *Cluster Index* is used during query evaluation, as shown in Figure 5.4.

The Spill Index is simply a cache containing information about which vertices in the graph are replicated across more than one G-by-Q cluster, the edge labels on such replicated vertices, the degrees of replicated vertices, the degree distribution of such replicated vertices across the G-by-Q clusters, and so on. This information is collected when query plan decisions are made and used in firing the conditional equivalence rules introduced in Section 5.3. Previous sections contained examples and discussions on how the Spill Index is utilized, therefore, further discussion is omitted in this section.

The Cluster Index is a collection of indexes built across the G-by-Q clusters and is utilized in step (1) of OptimalEvaluation after it is invoked from step (2) of SchemalessEvaluation (cf., Figure 5.1). Given a G-by-Q clustering $\mathbb{P}$ (of the RDF graph) that is being indexed and a query (or sub-query) Q, the Cluster Index is responsible for returning, *ideally*[3], the *prune* of $\mathbb{P}$ with respect to Q (cf., Definition 12).

There are three parts to the Cluster Index. Two simple graph indexes, namely, the Vertex and Edge indexes, maintain information about which particular URI maps to which G-by-Q cluster. The implementation of these indexes are trivial, hence, a more detailed

[3]This will be explained shortly.

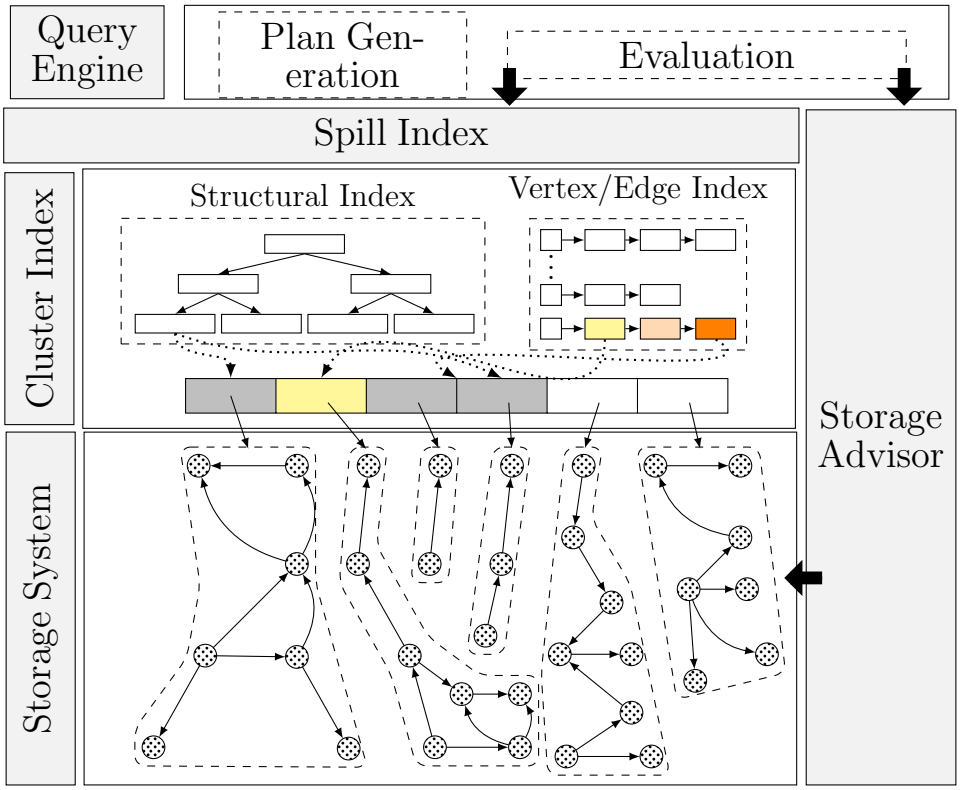


Figure 5.4: Detailed architecture of chameleon-db

discussion is omitted. The third part of the Cluster Index is consists of the Structural Index, whose details are discussed below.

An important property of the Structural Index is that it is built adaptively (with respect to the workload) and in a partial manner. In other words, the index is *not* built upfront, but incrementally, as queries are executed. Therefore, each query in the workload is treated as some advice on what to index in the next iteration—much like in database cracking [113–115].

The aforementioned adaptive, partial indexing scheme is implemented by relaxing the condition on what the index should return: Given a query, instead of strictly returning the prune of a G-by-Q clustering with respect to that query, the Structural Index is allowed to return additional clusters that are not in the prune set (i.e., false positives). These false positives are eliminated in a validation step later on during query evaluation. However, the Structural Index is never allowed to miss clusters that are in the prune set (i.e., false negatives). In other words, for any query Q and any G-by-Q clustering $\mathbb{P}$, $\sigma_Q^+(\mathbb{P}) \supseteq \sigma_Q(\mathbb{P})$

should hold, where $\sigma_Q^+(\mathbb{P})$ denotes the set of clusters that are returned by the Structural Index and $\sigma_Q(\mathbb{P})$ denotes the actual prune set (cf., Definition 12).

In the worst case, the Structural Index returns all of the clusters in the G-by-Q clustering of an RDF graph (i.e., $\mathbb{P}$), which means that queries need to be evaluated aggressively against the entire RDF graph, which may not be ideal from a performance point of view. On the other hand, the cost of building this index is minimal—in fact, simply maintaining a list of pointers to the G-by-Q clusters would be sufficient. The index can be made more efficient, for example, by sorting the list of pointers based on the number of edges in the referenced G-by-Q clusters. In that case, if it can be inferred that all matching subgraphs of a query (cf., Definition 5) need to have at least k edges, where k is some arbitrary constant, then, the Structural Index can return only those G-by-Q clusters that have k edges or more. This results in fewer false positives than the worst-case scenario, but it comes at the expense of additional indexing overhead.

The Structural Index in chameleon-db balances the trade-off between the indexing overhead and query performance by indexing only for queries it assumes will be present in the workload. This way, chameleon-db avoids wasting any effort on improving the Structural Index for queries that will never be executed, while making sure that queries that are actually part of the workload can be executed more efficiently. In this respect, it is assumed that queries that have been executed so far by the system are representative of the future queries in the workload (more sophisticated workload predictability assumptions, e.g., that can handle skew in workloads, are future work).

Before the implementation details of the Structural Index are presented, some notation and formalisms need to be introduced.

Definition 17. *Given a basic graph pattern $Q = (\hat{V}, \hat{E})$, its* generalized structural form *(GSF) is a basic graph pattern $Q^\diamond = (\hat{V}^\diamond, \hat{E}^\diamond)$ such that*

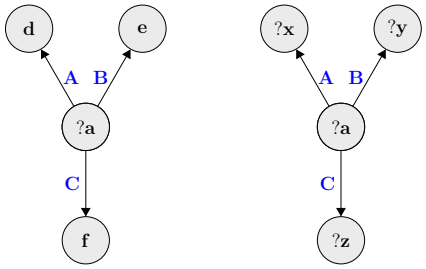
- $\hat{V}^\diamond = \big\{\, g(\hat{v}) \;\big|\; \hat{v} \in \hat{V} \big\}$, *and*
- $\hat{E}^\diamond = \big\{\, \big(\, g(\hat{s}),\, \hat{p},\, g(\hat{o})\big) \;\big|\; (\hat{s}, \hat{p}, \hat{o}) \in \hat{E} \big\}$.

where $g : (\mathcal{V} \cup \mathcal{U} \cup \mathcal{L}) \to \mathcal{V}$ is an injective function with the property that

$$g(\hat{v}) = \begin{cases} \hat{v} & : \hat{v} \in \mathcal{V} \\ \hat{v}^\diamond \in \mathcal{V} \setminus vars(\hat{E}) & : \hat{v} \in \mathcal{U} \cup \mathcal{L} \end{cases}$$

and $vars(\hat{E})$ is the set of variables mentioned in $\hat{E}$ such that

$$vars(\hat{E}) = \{\, x \;|\; \exists (\hat{s}, \hat{p}, \hat{o}) \in \hat{E} \;\; x \in \{\hat{s}, \hat{p}, \hat{o}\} \cap \mathcal{V} \}$$



(a) Q_A (b) $Q_A^{\diamond}$

Figure 5.5: Example of generalized structural form (GSF)

□

Example 7. *Consider the BGP in Figure 5.5a. It consists of four vertices. Among these four vertices, only one of them represents a variable, namely, $?a \in \mathcal{V}$. The remaining three are RDF terms (i.e., URIs or literals). The BGP in Figure 5.5b is the generalized structural form of Q_A. In translating Q_A to $Q_A^{\diamond}$, each vertex that represents an RDF term has been replaced with a distinct variable that is different from the set of variables mentioned in the (edge triples of the) original query, namely, the set $\{?a\}$.* □

Theorem 14. *Given a clustering $\mathbb{P}$ of an RDF graph and a BGP Q*

$$\sigma_{Q^{\diamond}}(\mathbb{P}) \supseteq \sigma_Q(\mathbb{P})$$

where $Q^{\diamond}$ is the generalized structural form of Q.

Proof 14. *To prove Theorem 14, it needs to be shown that for an RDF graph P, the following statement holds:*

$$\textit{If } P \in \sigma_Q(\mathbb{P}) \textit{ then } P \in \sigma_{Q^{\diamond}}(\mathbb{P}). \tag{5.11}$$

Equation 5.11 is proven by showing that $P \in \sigma_{Q^{\diamond}}(\mathbb{P})$ holds under the assumption that $P \in \sigma_Q(\mathbb{P})$ is true. To prove $P \in \sigma_{Q^{\diamond}}(\mathbb{P})$, proof-by-contradiction is used. Consequently, the proof steps are as follows:

S1. Let $vars(Q^{\diamond})$ denote the variables mentioned in $Q^{\diamond}$ and $vars(Q)$ denote the variables mentioned in Q (cf., Definition 17).

S2. *Let $g : (\mathcal{V} \cup \mathcal{U} \cup \mathcal{L}) \to \mathcal{V}$ denote the injective function that maps labels of vertices in Q to labels of vertices in $Q^\diamond$ (cf., Definition 17).*

S3. *Assume $P \in \sigma_Q(\mathbb{P})$.*

- *(a) Assume $P \notin \sigma_{Q^\diamond}(\mathbb{P})$.*
- *(b) According to Definition 12 and statement S3, $[\![Q]\!]_P \neq \emptyset$.*
- *(c) At the same time, according to Definition 12 and statement S3a, $[\![Q^\diamond]\!]_P = \emptyset$ holds because, otherwise, $P \in \sigma_{Q^\diamond}(\mathbb{P})$ has to be true.*
- *(d) According to Definition 5 and statement S3b, there exists a solution mapping μ and an RDF graph $P' = (V', E')$ such that P' is a subgraph of P and P' μ-matches Q.*
- *(e) According to Definition 4 and statement S3d, $dom(\mu) = vars(Q)$ and there exist two surjective functions $M_V : \hat{V} \to V'$ and $M_E : \hat{E} \to E'$ that satisfy the conditions in Definition 4.*
- *(f) Let $\mu^\diamond$ denote a solution mapping such that*
  - *i. $dom(\mu^\diamond) = vars(Q^\diamond)$;*
  - *ii. for every variable $?x \in dom(\mu^\diamond) \cap dom(\mu)$, $\mu^\diamond(?x) = \mu(?x)$; and*
  - *iii. for every variable $?y \in dom(\mu^\diamond) \setminus dom(\mu)$, $\mu^\diamond(?y) = g^{-1}(?y)$.*

  Note that since g is injective, its inverse g^{-1} is also injective; furthermore, by definition, g^{-1} is defined for every $?y \in vars(Q^\diamond) \setminus vars(Q)$ (cf., Definition 17), hence, for every $?y \in dom(\mu^\diamond) \setminus dom(\mu)$.
- *(g) Let $M_V^\diamond : \hat{V}^\diamond \to V'$ and $M_E^\diamond : \hat{E}^\diamond \to E'$ denote two surjective functions such that*
  - *i. for every vertex $\hat{v}^\diamond \in \hat{V}^\diamond$, $M_V^\diamond(\hat{v}^\diamond) = M_V(g^{-1}(\hat{v}^\diamond))$; and*
  - *ii. for every edge $\hat{e}^\diamond = (\hat{s}, \hat{p}, \hat{o}) \in \hat{E}^\diamond$, $M_E^\diamond(\hat{e}^\diamond) = M_E(\hat{e})$,*

  where $\hat{e} = (g^{-1}(\hat{s}), \hat{p}, g^{-1}(\hat{o}))$.
- *(h) According to Definition 4 and statements S3f and S3g, P' $\mu^\diamond$-matches $Q^\diamond$.*
- *(i) According to Definition 5 and statement S3h, $[\![P']\!]_{Q^\diamond} \neq \emptyset$, which contradicts statement S3c.*
- *(j) Thus, proof-by-contradiction suggests $P \in \sigma_{Q^\diamond}(\mathbb{P})$.*

S4. *Since statement S3j holds under the assumption $P \in \sigma_Q(\mathbb{P})$, the if statement in Equation 5.11 holds, thus, proving Theorem 14.* □

Theorem 15. *Given a group-by-query clustering $\mathbb{P}$ of an RDF graph and two BGPs $Q_A = (\hat{V}_A, \hat{E}_A)$ and $Q_B = (\hat{V}_B, \hat{E}_B)$ such that Q_1 is a strict supergraph of Q_2, then*

$$\sigma_{Q_A}(\mathbb{P}) \subseteq \sigma_{Q_B}(\mathbb{P}).$$

Proof 15. *To prove Theorem 15, it needs to be shown that for an RDF graph P, the following statement holds:*

$$\text{If } P \in \sigma_{Q_A}(\mathbb{P}) \text{ then } P \in \sigma_{Q_B}(\mathbb{P}). \tag{5.12}$$

Equation 5.12 is proven by showing that $P \in \sigma_{Q_B}(\mathbb{P})$ holds under the assumption that $P \in \sigma_{Q_A}(\mathbb{P})$ is true. To prove $P \in \sigma_{Q_B}(\mathbb{P})$, proof-by-contradiction is used. Consequently, the proof steps are as follows:

S1. Assume $P \in \sigma_{Q_A}(\mathbb{P})$.

- *(a) Assume $P \notin \sigma_{Q_B}(\mathbb{P})$.*
- *(b) According to Definition 12 and statement S1, $[\![Q_A]\!]_P \neq \emptyset$.*
- *(c) At the same time, according to Definition 12 and statement S1a, $[\![Q_B]\!]_P = \emptyset$ holds because, otherwise, $P \in \sigma_{Q_B}(\mathbb{P})$ has to be true.*
- *(d) Since Q_A is a strict supergraph of Q_B, there must exist a BGP $Q_\Delta = (\hat{V}_\Delta, \hat{E}_\Delta)$ such that Q_A is $Q_B \oplus Q_\Delta$ (cf., Definition 16).*
- *(e) According to Definition 5 and statement S1b, there exists a solution mapping μ_A and an RDF graph P_A such that P_A is a subgraph of P and P_A μ-matches Q_A.*
- *(f) Since Q_A is $Q_B \oplus Q_\Delta$, according to Lemma 9 and statement S1e, there exist two solution mappings μ_B and μ_Δ and two RDF graphs P_B and P_Δ such that*
  - *i. P_A is $P_B \oplus P_\Delta$,*
  - *ii. P_B μ_2-matches Q_B, and*
  - *iii. P_Δ μ_Δ-matches Q_Δ.*
- *(g) Then, according to Definition 5,*

$$[\![Q_B]\!]_{P_B} \neq \emptyset. \tag{5.13}$$

- *(h) Since P_B is a subgraph of P_1, and P_A is a subgraph of P, Equation 5.13 can be restated as $[\![Q_B]\!]_P \neq \emptyset$, which contradicts statement S1c.*
- *(i) Thus, proof-by-contradiction suggests $P \in \sigma_{Q_B}(\mathbb{P})$.*

S2. Since statement S1i holds under the assumption $P \in \sigma_{Q_A}(\mathbb{P})$, the if statement in Equation 5.12 holds, thus, proving Theorem 15. □

Initially, the Structural Index consists of just a list of pointers, referencing all of the G-by-Q clusters (i.e., $\mathbb{P}$). When the first query (Q_1) is executed, the whole list is traversed and all of the G-by-Q clusters are returned to the query engine. However, as the list is being traversed, additional computations are made and the index is reorganized (for subsequent queries). In particular, first, the query is generalized to a purely structural form (cf., Definition 17). Informally, what this means is that each RDF term that appears in the vertices of the query is replaced with a distinct variable.

In the second step, the list of pointers are partitioned around a pivot point such that pointers to G-by-Q clusters in $\sigma_{Q_1^\diamond}(\mathbb{P})$ are moved to the left of the pivot, and pointers to G-by-Q clusters that are *not* in $\sigma_{Q_1^\diamond}(\mathbb{P})$ remain at or to the right of the pivot, where $Q_1^\diamond$ denotes the GSF of Q_1. Let LEFT denote the access path to G-by-Q clusters on the left side of this pivot, and RIGHT the access path to the G-by-Q clusters that are at or on the right side of the pivot. The overhead of restructuring the list of pointers is small. Note that the list has to be traversed anyway to compute the matching subgraphs, and while doing so, pointers can be reordered in-place and in one-pass over the list—just like the partitioning step of the quicksort algorithm [65].

For the next query (Q_2), there are three possible scenarios. Let $Q_1^\diamond$ and $Q_2^\diamond$ denote the GSFs of Q_1 and Q_2, respectively.

- If the GSF of Q_2 is the same as the GSF of Q_1, the index can return the set of G-by-Q clusters that are reachable by the access path LEFT. Theorem 14 suggests that it is safe to do so, because

  $$\sigma_{Q_2}(\mathbb{P}) \subseteq \sigma_{Q_2^\diamond}(\mathbb{P}) \equiv \sigma_{Q_1^\diamond}(\mathbb{P})$$

  holds. G-by-Q clusters that are returned, but in which there is no matching subgraph of Q_2, can be discarded later on in the validation step.

- If the GSF of Q_2 is a strict supergraph of the GSF of Q_1, Theorem 14 and Theorem 15 suggest that

  $$\sigma_{Q_2}(\mathbb{P}) \subseteq \sigma_{Q_2^\diamond}(\mathbb{P}) \subseteq \sigma_{Q_1^\diamond}(\mathbb{P}).$$

  Therefore, the index can safely return the set of G-by-Q clusters that are reachable by the access path LEFT. However, since $\sigma_{Q_2^\diamond}(\mathbb{P}) \subset \sigma_{Q_1^\diamond}(\mathbb{P})$ can also be true, the

index needs to be reorganized again, this time for $Q_2^\diamond$. In particular, the pointers that are reachable by the access path LEFT are further partitioned into two segments LEFT $\rightarrow$ LEFT and LEFT $\rightarrow$ RIGHT such that the access path LEFT $\rightarrow$ LEFT leads to clusters that are in $\sigma_{Q_2^\diamond}(\mathbb{P})$ while the access path LEFT $\rightarrow$ RIGHT leads to clusters that are in $\sigma_{Q_1^\diamond}(\mathbb{P})$ but *not* in $\sigma_{Q_2^\diamond}(\mathbb{P})$.

- For all other cases, potentially some G-by-Q clusters that are accesible from both the LEFT and RIGHT access paths are in $\sigma_{Q_2^\diamond}(\mathbb{P})$. Therefore, both parts of the list (i.e., LEFT and RIGHT) need to be partitioned further, thus, resulting in four new access paths to be created: LEFT $\rightarrow$ LEFT, LEFT $\rightarrow$ RIGHT, RIGHT $\rightarrow$ LEFT, and RIGHT $\rightarrow$ RIGHT.

As the list of pointers gets divided into multiple partitions, the system maintains the access paths that are created using a binary search tree. The internal nodes of the tree store the generalized structural forms of the executed queries, while the leaf level contains pointers to the G-by-Q clusters. Given a query, the pseudocode of the algorithm for returning the set of G-by-Q clusters that are relevant to the query and, at the same time, updating the access paths and maintaining the binary search tree is given in Algorithm 15.

Given a query Q and its generalized structural form $Q^\diamond$, Algorithm 15 locates pointers to the relevant G-by-Q clusters and updates the index as follows: The tree is searched recursively in a top-down fashion. At every internal node of the tree, $Q^\diamond$ is compared against the query stored in the node's predicate. If $Q^\diamond$ is the same as the node's predicate, the left access path is taken and the tree is not updated. Otherwise, if $Q^\diamond$ is a strict supergraph of the query stored in the node's predicate, then, search proceeds recursively on the left access path, else it proceeds recursively on both access paths. When a leaf node is reached, the segment of pointers are divided into two partitions as described earlier, unless $Q^\diamond$ has been encountered previously in the recursion stack. If partitioning takes place, the tree is updated by making the leaf node an internal node and creating two leaf nodes, one for each partition. In the worst-case, GET-AND-MAINTAIN (cf., Algorithm 15) traverses and partitions the whole list of pointers to the G-by-Q clusters, therefore, it has $O(n)$ computational complexity (n denotes the number of G-by-Q clusters in the database).

To efficiently invalidate false-positive G-by-Q clusters that are returned by the Structural Index, second-level indexes are maintained. In particular, the *vertex-index* is a hashtable that maps URIs to the subset of G-by-Q clusters that contain the URI in their set of vertices, and the *range-index* keeps track of the minimum and maximum literal values within each G-by-Q cluster and acts as an additional filter. These two indexes also are built adaptively: Initially they are empty, but every time the Partition procedure (Algorithm 16) is called (i.e., for the construction of the Structural Index) the set of G-by-Q

clusters that are re-located to the left of the pivot are indexed in the vertex-index and the range-index.

Lastly, updates are discussed. The expectation is that the G-by-Q clusters are frequently updated in chameleon-db to accommodate changes in the workload even when the data are not updated. In the Structural Index, these updates are implemented as a combination of deletions and insertions. Deleting a G-by-Q cluster implies that the pointer referencing the G-by-Q cluster needs to be removed from the list, which has a trivial implementation and $O(1)$ computational complexity. On the other hand, inserting a G-by-Q cluster implies that the tree needs to be traversed top-down until a leaf-node is reached to locate the sub-list of pointers that need to be updated. Let P denote the G-by-Q cluster that is being inserted and let $Q_n^\diamond$ denote the predicate stored at the current node that is being traversed. Then, tree traversal proceeds with the left child of the current node if there is a subgraph in P that has a match for $Q_n^\diamond$, and with the right child otherwise. Since the decision taken at each node is a binary decision, search is deterministic and there exists exactly one access path to which P can be inserted. On average, insertion is an $O\big(log(m)\big)$ operation, where m denotes the number of nodes in the tree, but in the worst-case it could go up to $O(m)$ when the tree is not balanced (balancing the tree is a topic of future work).

Algorithm 15 GetAndMaintain

Require:

Q$^\diamond$: Query in generalized structural form

node: A node in the binary search tree

reOrganize: Boolean variable indicating whether it is necessary to re-organize the underlying list of pointers

Ensure:

Set of pointers to G-by-Q clusters for which $Q^\diamond$ has a match is returned and if necessary the list of pointers and the tree (rooted at node) is re-organized.

```
procedure GetAndMaintain(Q^◇, node, reOrganize)
    if node.isLeaf() then                                   ▷ Base case of recursion
        r ← MakeVector(node.begin, node.end)
        if reOrganize then
            pivot ← Partition(Q^◇, node.begin, node.end)          ▷ List is partitioned
            node.left ← MakeLeaf(node.begin, pivot −1)      ▷ Tree expanded to store
            node.right ← MakeLeaf(pivot, node.end)          ▷ information about partitions
            node.predicate ← Q^◇
            node.SetLeaf(false)
        end if
        return r
    else
        if Q^◇ == node.predicate then                       ▷ An equivalent query is encountered
            return GetAndMaintain(Q^◇, node.left, false)    ▷ No need to re-organize
        else if isSuperGraph(Q^◇, node.predicate) then
            return GetAndMaintain(Q^◇, node.left, reOrganize)
        else                                    ▷ Both left and right access paths may contain results
            r_1 ← GetAndMaintain(Q^◇, node.left, reOrganize)
            r_2 ← GetAndMaintain(Q^◇, node.right, reOrganize)
            return r_1 ∪ r_2
        end if
    end if
end procedure
```

Algorithm 16 Partition

Require:
Q$^\diamond$: Query in generalized structural form
i: Begin index (inclusive) on list of pointers to be partitioned
j: End index (inclusive) on list of pointers to be partitioned

Ensure:
Array of pointers are partitioned [65] such that pointers to G-by-Q clusters that have a matching subgraph of $Q^\diamond$ are placed before pointers to G-by-Q clusters that do not have a matching subgraph of $Q^\diamond$.

```
procedure PARTITION(Q^◇, i, j)
    pivot ← i
    cursor ← i
    for cursor ≤ j do
        if HASMATCH(Q^◇, *cursor) then
            SWAP(pivot, cursor)
            pivot++
        end if
        cursor++
    end for
    return pivot
end procedure
```

5.6 Discussion

Chapter 4 introduced the G-by-Q representation and showed its advantages using contrasting examples. Sections 4.2 and 4.3 introduced two practical algorithm to compute good G-by-Q representations. This chapter demonstrated that a poorly designed query evaluation algorithm could easily diminish the benefits of clustering, even if the clustering were to be perfect. Therefore, schemaless-evaluation was introduced, which is specifically optimized for the G-by-Q representation.

Schemaless-evaluation offers important advantages. First of all, it is possible to compute a clustering such that most of the queries in a workload do not require join operations across clusters. Even in the worst case when a query cannot be rewritten as any other expression, the baseline SE expression still guarantees correct results, thereby providing flexibility to compute a clustering that favors the most frequent queries in a workload and allowing the clustering to be updated as these frequencies change.

Second, the scope of subgraph matching is limited to contents within each cluster, thereby providing isolation with significant benefits. Since clusters are now truly isolated from each other, new clusters can be added and existing clusters can be split or merged without affecting the integrity of query evaluation on other parts of the graph. Consequently, query evaluation can be more easily interleaved with re-clustering of the graph, which is one of the key objectives of this work. There is also an opportunity for parallelization—subgraph matching can be performed concurrently on multiple clusters.

Third, when determining whether or not a cluster contains a subgraph that matches a query, the indexes need to consider only the subgraphs that reside within a single cluster. This reduction in search space reduces the indexing overhead and facilitates easy updating of the indexes.

Chapter 6

Evaluation

In this chapter, the techniques proposed in this thesis are experimentally evaluated. For the evaluations, the Waterloo SPARQL Diversity Test Suite (WatDiv) [18] is used. Section 6.1 discusses problems with existing SPARQL benchmarks and the reason why WatDiv was developed (and characteristics of WatDiv), and Section 6.2 discusses the results of the experimental evaluations using WatDiv.

6.1 Waterloo SPARQL Diversity Test Suite (WatDiv)

As argued earlier in Chapter 1, queries executed on RDF data management systems have become increasingly more diverse [30], [71], [124]. Existing systems have started to display unpredictable behaviour over these workloads, even to the extent that on some queries they time out. At the same time, data that are handled by these RDF data management systems have become far more heterogeneous [71], and web applications that are supported by these systems have become far more varied [30], [124]. Unfortunately, existing benchmarks do not have the corresponding variability in their datasets and workloads to reveal the true behavior of existing systems. Consequently, problems go undetected in evaluations using these benchmarks until systems are actually deployed. To address these shortcomings, the Waterloo SPARQL Diversity Test Suite (WatDiv) has been designed as part of this thesis that offers stress testing tools to reveal a much wider range of problems with RDF data management systems. Consequently, WatDiv is used across the experimental evaluations in this thesis.

The contributions with WatDiv and the work leading up to its design can be summarized in two steps. First, two classes of query features are introduced (cf., Section 6.1.1), namely,

structural and data-driven features that should be used to evaluate the variability of the datasets and workloads in a SPARQL benchmark. More specifically, these features aim to differentiate as much as possible those types of queries that may result in unpredictable system behaviour and are indicators of potential flaws in physical design. For example, previous work has illustrated that the choice of physical design in an RDF system is very sensitive to the types of joins that the system can efficiently support [19]. Hence, a feature called "join vertex type" is introduced. Likewise, a system's performance depends on the characteristics of the data as much as the query itself. Consequently, additional features that capture multiple notions of selectivity and result cardinality are introduced.

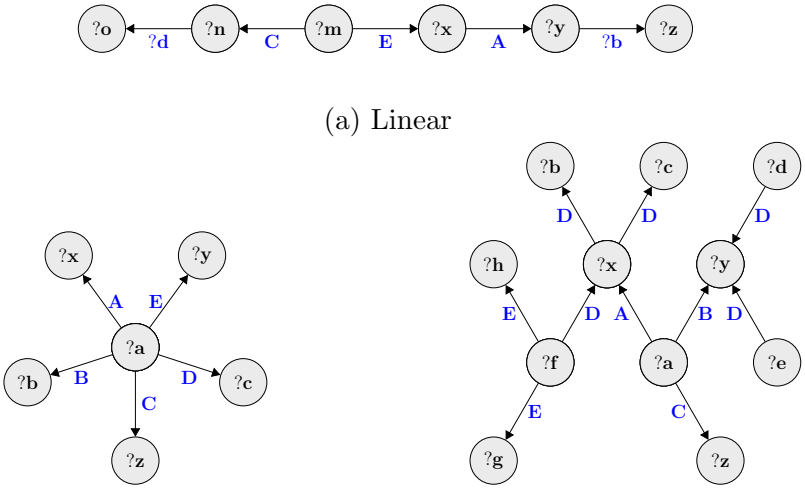
Second, an in-depth analysis is performed of existing SPARQL benchmarks using the two classes of query features that are introduced in this chapter. The experimental evaluation demonstrates that no single benchmark (including those that are based on actual query logs) is sufficiently varied to test whether a system has consistently good performance across diverse workloads (cf., Section 6.1.2). Furthermore, these benchmarks do not provide the tools to localize problems to specific types of queries if needed. For example, it would be useful if one could diagnose that the system under test has problems with queries that have a particular join vertex type, and drill down the evaluation if necessary. These are exactly the type of evaluations that are aimed to be facilitated with WatDiv.

This section is organized as follows. Section 6.1.1 introduces the features used in the evaluation of existing SPARQL benchmarks, and Section 6.1.2 discusses the results of these evaluations. Section 6.1.3 introduces WatDiv while Section 6.1.4 compares WatDiv to existing benchmarks with respect to the aforementioned features.

6.1.1 Preliminaries

This section defines the query features based on which the diversity of SPARQL benchmark workloads can be discussed. These features can be categorized into two classes: (i) structural features and (ii) data-driven features [18].

These features are defined over a basic fragment of SPARQL – BGPs with filter expressions – which sufficiently covers the queries within the scope of this thesis. For the sake of brevity, the queries in this fragment are denoted by a pair $\bar{B} = \langle B, F \rangle$, hereafter, referred to as a *constrained BGP* (*CBGP*), where B is a finite set of triple patterns (i.e., a BGP) and F is a finite set of SPARQL filter expressions. Hence, by using the algebraic syntax for SPARQL [28], a CBGP $\bar{B} = \langle B, F \rangle$ with $F = \{f_1, \ldots, f_n\}$ is equivalent to a SPARQL graph pattern P of the form $\big((\ldots(B \text{ FILTER } f_1)\ldots) \text{ FILTER } f_n\big)$ (if $F = \emptyset$, then P is the BGP B). Consequently, the *evaluation* of $\bar{B}$ over an RDF graph G, denoted by



(a) Linear

(b) Star

(c) Snowflake

Figure 6.1: Example Query Structures

$[\![\bar{B}]\!]_G$, is the set of solution mappings $[\![P]\!]_G$ as defined by the standard SPARQL query semantics [28], [99].

Structural Features

Every BGP (as used by a CBGP) combines a set of triple patterns into numerous query structures such as those in Figures 6.1a–6.1c. As a basis for comparing the structural diversity of different sets of CBGPs, four features are used in this thesis.

- **Triple Pattern Count**: This feature refers to the number of triple patterns in (the BGP of) a CBGP. Triple pattern count allows one to broadly distinguish between simple and structurally complex queries. Ideally, one would like an RDF data management system to execute simple queries extremely fast while scaling well with increasing number of triple patterns. In fact, DBpedia query logs [143] reveal that while in general most queries contain only a few triple patterns, users may issue (albeit infrequently) queries having more than 50 triple patterns.
- **Join Vertex Count**: This feature represents the number of RDF terms (i.e., URIs and literals) and variables that are the subject or object of multiple triple patterns in a CBGP. Hereafter, these terms and variables are referred to as the *join vertices*

of the CBGP. Formally, if $\mathcal{U}$, $\mathcal{L}$ and $\mathcal{V}$ denote the set of all URIs, the set of all literals and the set of all variables, respectively, then an element $x \in (\mathcal{U} \cup \mathcal{L} \cup \mathcal{V})$ is a *join vertex* of CBGP $\bar{B} = \langle B, F \rangle$ if there exist two distinct triple patterns $tp = (s, p, o)$ and $tp' = (s', p', o')$ such that

  - $tp \in B$ and $tp' \in B$,
  - $x \in \{s, o\}$, and
  - $x \in \{s', o'\}$.

- **Join Vertex Degree**: For each join vertex x of a CBGP $\bar{B} = \langle B, F \rangle$, the *degree* of x is the number of triple patterns in B whose subject or object is x. Hereafter, for any such triple pattern $(s, p, o) \in B$ with $x \in \{s, o\}$, the triple pattern is said to be *incident* on x.

  Join vertex count and join vertex degree offer a finer distinction of structural complexity than the triple pattern count. For example, the two queries in Figure 6.1a and Figure 6.1b have the same number of triple patterns but they differ in their join vertex count and join vertex degrees. That is, Figure 6.1a is a long linear-shaped query with multiple (4) low-degree (2) join vertices, whereas Figure 6.1b is a star-shaped query with a single high-degree (5) join vertex. A system may show completely different performance for these two queries and a stress-testing benchmark should capture such blind spots if any.

- **Join Vertex Types**: The data representation and indexing schemes employed by RDF systems can result in completely different behaviour on different types of joins [19], and a benchmark should include a sufficiently large sample of queries for each join type. Consequently, the following three (mutually non-exclusive) types of join vertices are distinguished based on the types of joins that will need to be executed during query evaluation: A join vertex x of a CBGP $\bar{B} = \langle B, F \rangle$ is of type

  - SS^+ if $x = s$ for every triple pattern $(s, p, o) \in B$ that is incident on x;
  - OO^+ if $x = o$ for every $(s, p, o) \in B$ that is incident on x; and
  - SO^+ if $x = s$ and $x = o'$ for two triple patterns $(s, p, o) \in B$ and $(s', p', o') \in B$ (both of which are incident on x, respectively).

  For example, the join vertices $?a$, $?x$ and $?y$ in Figure 6.1c have types SS^+, SO^+ and OO^+, respectively.

Data-driven Features

The structural query features (discussed above) are often not sufficient. More specifically, a system's choice of a query (execution) plan depends on the characteristics of the data as much as the query itself. For example, systems rely heavily on selectivity and cardinality estimations for query plan optimization [176]. Consider the following example: A system chooses to break down a BGP $B = \{tp_A, tp_B, tp_C\}$ into its triple patterns and to execute them in a specific order, namely, tp_A, tp_B and then tp_C. The system picks this particular query plan because the subset of triples that match tp_A is smaller. Furthermore, it estimates the intermediate result cardinalities to be sufficiently low and decides to use in-memory data structures and algorithms. To measure how diversely a benchmark covers systems' different plan choices such as the one above, the following test cases are considered:

- queries have a diverse mix of result cardinalities;
- a single or few triple patterns are very selective, while the remaining ones are not;
- all of the triple patterns in a query are almost equally selective (hence, there is a higher probability that the optimizer picks a sub-optimal query plan due to estimation errors); etc.

Next, various notions of result cardinality and selectivity are defined to distinguish among the aforementioned test cases.

- **Result Cardinality**: This feature represents the number of solutions in the result of evaluating a CBGP $\bar{B} = \langle B, F \rangle$ over an RDF graph G. Recall that this result, denoted by $[\![\bar{B}]\!]_G$, is a set of solution mappings (cf. Section 6.1.1). Consequently, if Ω denotes the set $[\![\bar{B}]\!]_G$ and $\text{card}_{[\![\bar{B}]\!]_G}$ denotes the function that maps each solution mapping $\mu \in \Omega$ to its cardinality in the set [28], the result cardinality of $\bar{B}$ over G is defined as

$$\textsc{card}(\bar{B}, G) = \sum_{\mu \in \Omega} \text{card}_{[\![\bar{B}]\!]_G}(\mu). \tag{6.1}$$

- **Filtered Triple Pattern Selectivity (f-TP Selectivity)**: Given some CBGP $\bar{B} = \langle B, F \rangle$ and a BGP B^* such that $B^* \subseteq B$, let $\lambda^F(B^*)$ denote the CBGP $\bar{B}' = \langle B', F' \rangle$ with $B' = B^*$ and $F' = \{f \in F \mid \text{vars}(f) \subseteq \text{vars}(B^*)\}$, where $\text{vars}(\cdot)$ denotes the variables in a filter expression or a BGP. Then, for any triple pattern $tp \in B$ in

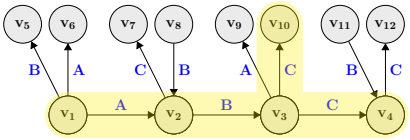
a CBGP $\bar{B} = \langle B, F \rangle$, the *f-TP selectivity* of tp over an RDF graph G, denoted by $\text{SEL}_G^F(tp)$, is the ratio of distinct solution mappings in $[\![\lambda^F(\{tp\})]\!]_G$ to the number of triples in G. Formally, if Ω denotes the query result set $[\![\lambda^F(\{tp\})]\!]_G$, then

$$\text{SEL}_G^F(tp) = \frac{|\Omega|}{|G|}. \tag{6.2}$$

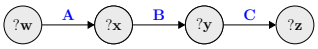
In evaluations of existing benchmarks, three related measures are used. The result cardinality of a CBGP is used as it is defined. For f-TP selectivities, the *mean* and *standard deviation* of the f-TP selectivities of the triple patterns in the CBGP are reported. The latter is especially important in distinguishing queries whose triple patterns are almost equally selective from queries with varying f-TP selectivities.

While result cardinality and f-TP selectivity are useful features, they are not sufficient. More specifically, once a system picks a particular query plan and starts executing it, it is often the case that there are intermediate solution mappings which do not make it to the final result. What this means is that *all* triple patterns of a CBGP contribute to its overall "selectiveness", or stated differently, in every join step, some intermediate solution mappings are being pruned. Contrast this to another possible case in which the overall "selectiveness" of a CBGP can be attributed to a single triple pattern in that CBGP. In that case, a system could use runtime optimization techniques such as sideways-information passing [147] to *early-prune* most of the intermediate results, which may not be possible in the original example (for a more detailed discussion refer to [19]). From a testing point of view, it is important to include both cases. In fact, in Section 6.2, this example is revisited and it is experimentally shown that systems behave differently on these two cases. To capture these constraints, two more features are used, namely BGP-restricted and join-restricted f-TP selectivity. The former is concerned with how much a filtered triple pattern contributes to the overall "selectiveness" of the query, whereas the latter is concerned with how much a filtered triple pattern contributes to the overall "selectiveness" of the join(s) that it participates in. Just as is done for f-TP selectivity, the mean and standard deviation of these two features are reported in the evaluations of benchmarks.

- **BGP-Restricted f-TP Selectivity**: For any triple pattern $tp \in B$ in a CBGP $\bar{B} = \langle B, F \rangle$, the *$\bar{B}$-restricted f-TP selectivity* of tp over an RDF graph G, which is denoted by $\text{SEL}_G^F(tp \mid \bar{B})$, is the fraction of distinct solution mappings in $[\![\lambda^F(\{tp\})]\!]_G$ that are *compatible* (as per standard SPARQL semantics [28]) with a solution mapping in the query result $[\![\bar{B}]\!]_G$. Formally, if Ω and Ω' denote the query result sets $[\![\lambda^F(\{tp\})]\!]_G$



(a) Sample RDF Graph



(b) Sample BGP

?w	?x
v_1	v_2
v_1	v_6
v_3	v_9

$\bowtie$

?x	?y
v_1	v_5
v_2	v_3
v_8	v_2
v_{11}	v_4

$\bowtie$

?y	?z
v_2	v_7
v_3	v_4
v_3	v_{10}
v_4	v_{12}

$=$

?w	?x	?y	?z
v_1	v_2	v_3	v_4
v_1	v_2	v_3	v_{10}

(c) Evaluation of BGP

Figure 6.2: Sample Evaluation

and $[\![\bar{B}]\!]_G$, respectively, then

$$\text{SEL}_G^F(tp \mid \bar{B}) = \frac{\big|\{\, \mu \in \Omega \mid \exists \mu' \in \Omega' :\ \mu \text{ and } \mu' \text{ are compatible} \,\}\big|}{\big|\Omega\big|}. \tag{6.3}$$

- **Join-Restricted f-TP Selectivity**: Given a CBGP $\bar{B} = \langle B, F \rangle$, a join vertex x of $\bar{B}$, and a triple pattern $tp \in B$ that is incident on x, the *x-restricted f-TP selectivity* of tp over an RDF graph G, denoted by $\text{SEL}_G^F(tp \mid x)$, is the fraction of distinct solution mappings in $[\![\lambda^F(\{tp\})]\!]_G$ that are compatible with a solution mapping in the (join) query result $[\![\lambda^F(B^x)]\!]_G$ with $B^x \subseteq B$ being the subset of all the triple patterns in B that are incident on x (i.e, $B^x = \{tp \in B \mid tp \text{ is incident on } x\}$). Hence, if Ω and Ω' denote the sets $[\![\lambda^F(\{tp\})]\!]_G$ and $[\![\lambda^F(B^x)]\!]_G$, respectively, then

$$\text{SEL}_G^F(tp \mid x) = \frac{\big|\{\mu \in \Omega \mid \exists \mu' \in \Omega' :\ \mu \text{ and } \mu' \text{ are compatible} \,\}\big|}{\big|\Omega\big|}. \tag{6.4}$$

Consider the RDF graph in Figure 6.2a and the BGP in Figure 6.2b. Consider the triple pattern $(?w, A, ?x)$ of the BGP. The BGP-Restricted f-TP selectivity of this triple

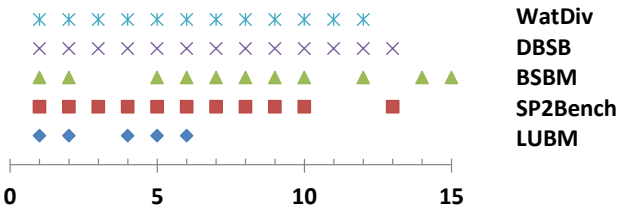
pattern is $\frac{1}{3}$. The reason is as follows: Note that the evaluation of this triple pattern against the RDF graph returns 3 solution mappings (i.e., tuples). Hence, the denominator in Equation 6.3 is 3. On the other hand, only 1 of these solution mappings is compatible with any of the solution mappings in the result of the evaluation of the BGP (highlighted in orange). Therefore, the numerator of Equation 6.3 is 1, thus, the BGP-Restricted f-TP selectivity is $\frac{1}{3}$.

6.1.2 Experimental Evaluation of Existing Benchmarks

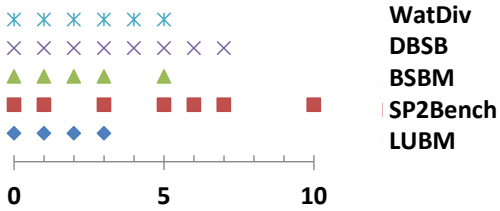
In this section, some of the existing SPARQL benchmarks [44], [91], [143], [167] are experimentally evaluated using the structural and data-driven features introduced in Section 6.1.1. This study demonstrates that these benchmarks lack diversity, hence, they fail to appropriately stress test RDF data management systems. In this study, the following more popular benchmarks are considered:

- The *Lehigh University Benchmark* (LUBM) [91] was originally designed for testing the inferencing capabilities of Semantic Web repositories.
- The *Berlin SPARQL Benchmark* (BSBM) [44] contains multiple use cases such as (i) explore, (ii) update, and (iii) business intelligence use cases. The explore use case is developed around an e-commerce scenario, where queries mimic the search patterns of consumers who are browsing through products. The update use case focuses on testing systems' support for updates, while the business intelligence use case is developed around an OLAP scenario. BSBM also goes into testing how well RDF systems support different (and important) SPARQL features, namely, aggregation, union, and optional graph patterns.
- *SP²Bench* [167] tests various SPARQL features such as union and optional graph patterns.
- The *DBpedia SPARQL Benchmark* [143] (DBSB) uses queries that have been generated by mining actual query logs over the DBpedia dataset [35]. Thus, it contains a more "diverse set of queries" [143].

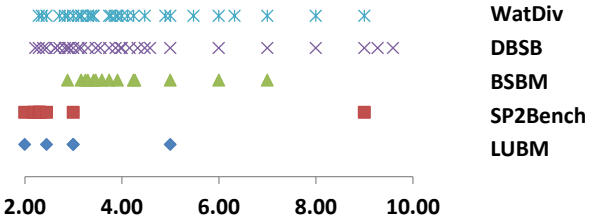
In the study reported here, only SELECT queries are considered. For BSBM, the study focuses on the explore use case, for which 100 queries have been generated for each query template. This has been observed to be a sufficiently large sample to understand the general properties of BSBM. For DBSB, a sample of 12500 queries have been drawn uniformly at random from the subset of SELECT queries in the query logs [143] (the other two benchmarks have a fixed number of queries). For WatDiv, the same number of queries (12500) have been generated.



(a) Triple pattern count



(b) Join vertex count



(c) Join vertex degree (mean)

	SS$^+$	OO$^+$	SO$^+$
LUBM	78.6%	0.0%	42.9%
SP^{2}Bench	81.0%	33.3%	57.1%
BSBM	84.8%	5.6%	52.8%
DBSB	41.1%	4.4%	5.4%
WatDiv	61.3%	18.0%	61.3%

(d) Distribution w.r.t. join vertex types

Figure 6.3: Analysis w.r.t. structural features: in Fig. 6.3a–6.3c, each point indicates the presence of a query with the corresponding x-axis value for a given feature.

Recall that the query features in Section 6.1.1 are defined over CBGPs. For this reason, when analyzing existing benchmarks (with respect to these features), first each complex non-CBGP query is translated into a CBGP by replacing OPT and UNION operators with AND. Hereafter, these CBGPs (including those for which translation was not necessary) are referred to as the queries of the benchmark. To compute the statistics reported in this section, for each benchmark, a benchmark-specific dataset of 1 million triples was generated, and queries of the benchmark were executed on this dataset.

Evaluation Using Structural Features

Consider Figure 6.3a, which compares queries in each benchmark with respect to their triple pattern count (x-axis).[1] Benchmarks are stacked along the y-axis. For each benchmark, the presence of a point indicates that the benchmark contains at least one query with the corresponding number of triple patterns indicated by the x-axis value. Figure 6.3a–6.3c and Figure 6.4a– 6.4f should be read similarly.

While most of the evaluated benchmarks contain large queries with more than 10 triple

[1]For the time being ignore WatDiv in these figures. The results about WatDiv are not important for this section. They will be discussed in Section 6.1.3.

patterns (cf., Figure 6.3a)[2], LUBM contains only small queries—not exceeding 6 triple patterns in cardinality. Furthermore, LUBM's join vertex count is also lower than the other benchmarks (Figure 6.3b). This is reasonable as LUBM is intended for semantic inferencing. In fact, the true complexity of an LUBM query lies in its semantics, not in its structure. For this reason, the suitability of LUBM for performance evaluation is limited if the system under test does not support inferencing.

By considering mean join vertex degrees (Figure 6.3c), it can be observed that DBSB is more diverse than any of the synthetic benchmarks (i.e., LUBM, BSBM, SP^{2}Bench). LUBM contains fairly simple queries (cf., Figure 6.3a), which explains why the mean join vertex degree is also low for most of these queries. SP^{2}Bench contains (i) linear queries that are long, or (ii) star queries that are large and centered around a single join vertex, but not much in between; hence, the join-vertex degree values are concentrated at the two ends of the x-axis in Figure 6.3c. BSBM contains queries that are a little bit more diverse in their join vertex degrees, but it does not test the two extremes as SP^{2}Bench does.

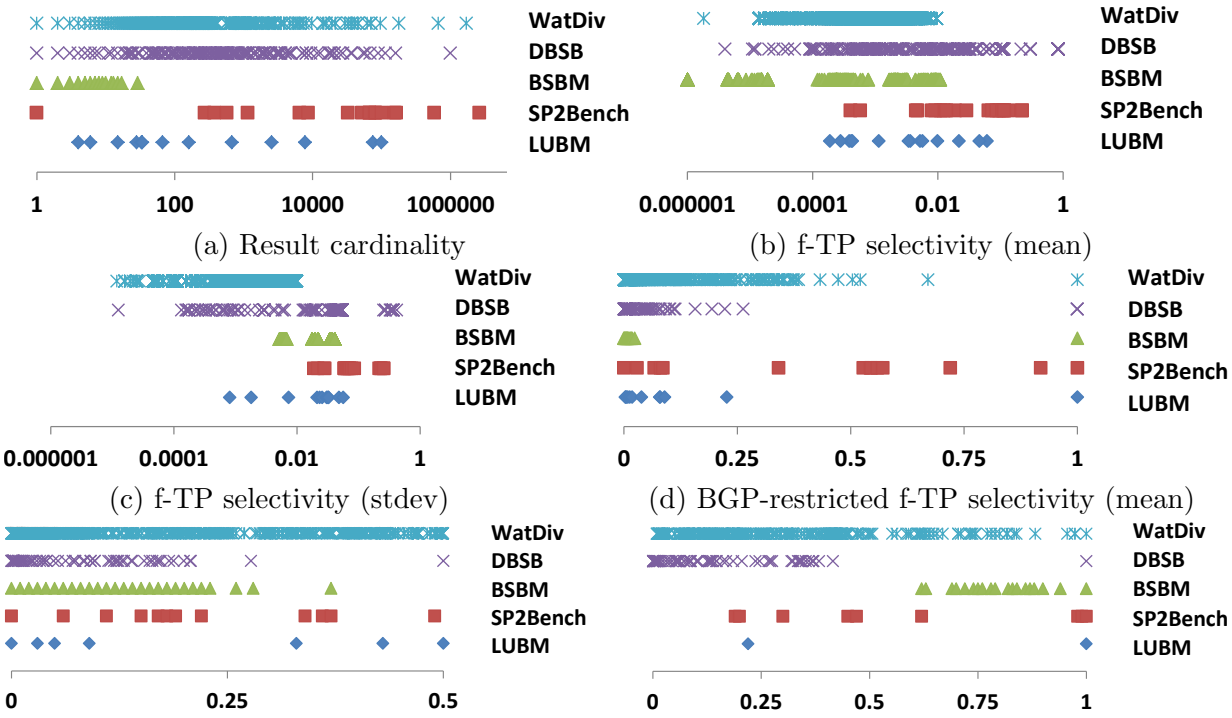
In Figure 6.3d, benchmarks are compared and contrasted with respect to the types of join vertices present in each of the queries. This comparison reveals three problems: LUBM does not contain any query with an OO$^+$ join; BSBM contains some, but their percentage is significantly low. In DBSB, queries with both OO$^+$ and SO$^+$ joins have a low percentage. Consequently, these three benchmarks may be biased towards particular physical designs that are more effective for SS$^+$ (or SO$^+$) joins, which limits the suitability of these benchmarks for stress tests.

Evaluation Using Data-Driven Features

Regarding result cardinality, the following observations can be made. BSBM contains only low-cardinality queries, SP^{2}Bench contains almost only high-cardinality queries, and LUBM contains only medium-cardinality queries (cf., Figure 6.4a), which reveals another limitation of what each of these three benchmarks can test individually.

Figure 6.4b–6.4c show another issue with the evaluated benchmarks. Although these benchmarks are fairly diverse with respect to f-TP selectivity (i.e., especially DBSB and BSBM), the standard deviation of f-TP selectivities of filtered triple patterns (within any single query) is generally high. As explained in Section 6.1.1, this implies that these benchmarks are missing the test case in which the triple patterns are more or less equally selective.

[2]Some DBSB queries have as many as 50 triple patterns, but for clarity of presentation, they are not displayed.



(a) Result cardinality

(b) f-TP selectivity (mean)

(c) f-TP selectivity (stdev)

(d) BGP-restricted f-TP selectivity (mean)

(e) BGP-restricted f-TP selectivity (stdev)

(f) Join-restricted f-TP selectivity (mean)

Figure 6.4: Analysis w.r.t. data-driven features at 1 million triples: each point indicates the presence of a query with the corresponding x-axis value for a given feature.

As depicted in Figure 6.4d, among the four benchmarks, only SP^{2}Bench has a diverse selection of queries regarding mean BGP-restricted f-TP. LUBM, BSBM and DBSB have queries in which either the mean value is 1.0, indicating that each triple pattern in separation does not contribute to the selectiveness of the query, or the mean is extremely low, indicating the opposite. For BSBM, the contrast is even more extreme. Figure 6.4e highlights an even further problem with DBSB and BSBM. For these two benchmarks, the variation in BGP-restricted f-TP lies mostly in the lower end of the spectrum, which indicates that these benchmarks cannot be used to test with queries in which triple patterns contribute unevenly to the pruning of intermediate results (cf., Section 6.1.1).

Finally, consider Figure 6.4f, which compares benchmark queries using join-restricted f-TP (mean). One can observe two important limitations. First, both LUBM and SP^{2}Bench queries sparsely cover the spectrum of possible values. Second, although BSBM and DBSB are much more diverse, they cover completely different ends of the spectrum. A system can generate completely different query plans for these two scenarios, and therefore, stress

testing should use workloads that include both scenarios.

In summary, the best known benchmarks (including DBSB, which is based on actual query logs), individually, are not sufficiently diverse to test the strengths and weaknesses of different physical design choices employed by RDF systems. Aggregating results from multiple benchmarks is not a good solution to the diversity problem either. First, the underlying datasets have completely different characteristics; therefore, one may get queries with completely disjoint distributions across the structural and data-driven features. For example, even though it may appear, based on Figure 6.4f, that DBSB and BSBM complement each other (i.e., they cover the opposite ends of the set of possible x-axis values), Figure 6.4a suggests that it is not quite so. The problem is that these two benchmarks do not complement each other on *all* possible features. Hence, in an aggregated (hypothetical) benchmark, one would still be missing queries with *high* cardinality and *high* join-restricted f-TP selectivity values. Second, scalability is an issue. It is not clear (i) how one can generate more queries given that some of the above-mentioned benchmarks have a fixed number of queries, or (ii) how results from multiple benchmarks should be combined given that each benchmark has its own scalability restrictions. WatDiv, which is introduced in the next section, is designed to address these issues.

6.1.3 Characteristics of WatDiv

WatDiv consists of multiple tools [22] that enable stress testing of RDF data management systems:

- The *data generator* generates scalable datasets at user-specified scale factors—a common feature of benchmarks. A more interesting feature is that data are generated according to the WatDiv schema[3] with customizable value distributions. A tutorial is available for customizing WatDiv's schema [16].
- The *query template generator* traverses the WatDiv schema and generates a diverse set of query templates (which is the first step in generating a workload for the stress tests). Users can specify the number of query templates to be generated as well as certain restrictions on the query templates such as the maximum number of triple patterns or whether predicates in triple patterns should be bound.
- Given a set of query templates, the *query generator* instantiates these templates with actual RDF terms from the dataset (which is the second and last step in generating

[3] http://db.uwaterloo.ca/watdiv/watdiv-data-model.txt

Entity Sets	Instances	
Purchase	1500	per scale factor
User	1000	
Offer	900	
Product	250	
Website	50	
Retailer	12	
Topic	250	constant
City	240	
SubGenre	145	
Language	25	
Country	25	
Genre	21	
ProductCategory	15	
AgeGroup	9	
Role	3	
Gender	2	

Figure 6.5: Entities generated according to the default WatDiv schema.

a workload for the stress tests). The number of queries to be instantiated per query template can be specified by users.

- Given a workload of BGP queries, the *query analysis tool* lets one analyze the diversity in the workload with respect to the query features discussed in Section 6.1.

Dataset Description

Table 6.5 lists the entities that are generated using the default WatDiv schema, and Table 6.6 lists the characteristics of the RDF dataset. The generated database is a typical electronic commerce database with a social-network component. That is, the database maintains information about products, retailers and users, where retailers can make offers on products, and users can purchase products and write reviews. Users are represented within a social-network, where users have friends and may follow each other. While a majority of the users are consumers, some users are producers as well—for example, a user might as well be the director of a movie, or the author of a book.

	Count
triples	105257
URIs	5947
literals	14286
distinct subjects	5597
distinct predicates	85
distinct objects	13258
distinct literals	8018

Figure 6.6: Characteristics of the dataset generated from the default WatDiv schema at scale factor= 1.

What distinguishes WatDiv datasets from existing RDF benchmarks is the diversity of the structuredness: some entity sets in WatDiv are well-structured, meaning that they contain few optional attributes, while some others are less well-structured [71]. As discussed in Section 6.1.4, this enables the generation of test queries that are far more diverse in their data-driven features.

Three properties contribute to the WatDiv's diversity. First, instances of the same entity set (i.e., class) do not necessarily have the same attributes. Consider the different types of entities used in WatDiv. *Product* instances may be associated with different *Product Categories* (e.g., Book, Movie, Classical Music Concert, etc.), but depending on the category a product belongs to, it will have a different set of attributes. For example, products that belong to the category "Classical Music Concert" have the attributes *opus*, *movement*, *composer* and *performer* (in addition to the attributes that are common to every product), whereas products that belong to the category "Book" have the attributes *isbn*, *bookEdition* and *numberOfPages*.

Second, even within a single product category, not all instances share the same set of attributes. For example, while *isbn* is a mandatory attribute for books, *bookEdition* ($Pr = 0.5$) and *numberOfPages* ($Pr = 0.25$) are optional attributes, where Pr indicates the probability that an instance will be generated with that attribute. Users are able to modify the WatDiv schema, hence these probabilities.

Third, a group of attributes can be correlated, which means that either all or none of the correlated attributes in that group will be present in any instance of the entity type. For example, *opus* and *movement* are two correlated attributes for "Classical Music Concert" products (cf. <pgroup> construct in the WatDiv dataset schema).

```
#mapping v1 wsdbm:Topic uniform
#mapping v6 wsdbm:ProductCategory normal
SELECT ?v0 ?v2 ?v3 ?v4 ?v5 WHERE {
   ?v0 og:tag %v1% .
   ?v0 rdf:type ?v2 .
   ?v3 sorg:trailer ?v4 .
   ?v3 sorg:keywords ?v5 .
   ?v3 wsdbm:hasGenre ?v0 .
   ?v3 rdf:type %v6% . }
```

Figure 6.7: Sample query template generated by WatDiv as a result of the random walk over the schema graph.

Test Queries

The benchmark queries are generated in two phases. First, a set of query templates are created by performing a random walk over the graph representation of the schema of the dataset (i.e., query template generator). In this regard, the following (internal) representation is used: every entity type in the schema corresponds to a graph vertex, relationships among entity types (i.e., which correspond to RDF predicates in the instantiated dataset) are represented using graph edges, and each vertex is annotated with the set of properties of that entity type. The traversal produces a set of BGPs with a maximum n triple patterns, where n was set to 15 in the experiments in Section 6.1. Note that BGPs are generated with triple patterns that have unbound subjects and objects, whereas their predicates are bound. Then, k uniformly randomly selected subjects/objects are replaced with WatDiv-specific placeholders (i.e., placeholders are enclosed within percentage [%] signs in the benchmark). In the second phase, placeholders in each query template are instantiated with RDF terms from the WatDiv dataset (i.e., query generator). To this end, the WatDiv tools maintain, for each placeholder, a set of values that are applicable to that placeholder, and during the instantiation phase, a value is drawn uniformly at random. For the study in Section 6.1, 12500 test queries have been generated from a total of 125 query templates (i.e., the same number of queries that were sampled in DBSB). These queries are available online.[4]

A query template that is generated during the first phase may look like the sample in Figure 6.7. This query template consists of six triple patterns. There are two placeholders in the template, which should be instantiated with actual RDF terms in the second

[4] http://db.uwaterloo.ca/watdiv/stress-workloads.tar.gz

phase. The first two lines in the query template indicate how the placeholders should be instantiated. The first line tells the WatDiv query generator that the placeholder $v1$ should be instantiated with instances of the *Topic* entity set. Furthermore, it tells the query generator that the values of $v1$ should be drawn from a uniform distribution. The second line tells the query generator that the placeholder $v6$ should be instantiated with *Product Categories*. However, this time, the values are drawn from a normal distribution.

6.1.4 Discussion

In Figures 6.3a–6.3d and Figures 6.4a–6.4f, the aforementioned 12500 WatDiv test queries are characterized. With respect to most of the structural query features, WatDiv has comparable characteristics to DBSB and it is far more diverse than LUBM, SP^{2}Bench and BSBM (cf., Figures 6.3a–6.3c). For example, the mean join vertex degree values are densely distributed between 2.0 and 10.0, indicating a rich mix of queries. Furthermore, with respect to join vertex types, WatDiv has a much more balanced distribution than DBSB: a significant 18.0% of queries in the WatDiv workload have OO$^+$-type join vertices, compared to only 4.4% in DBSB, and 61.3% versus 5.4% for queries with SO$^+$ joins.

With respect to most of the data-driven features, WatDiv is far more diverse, often filling in the gaps that are not supported by existing benchmarks (cf., Figure 6.4d, 6.4e and 6.4f). For example, while DBSB and BSBM cover only the opposite ends of the spectrum of mean join-restricted f-TP selectivity values, WatDiv covers the full spectrum (cf., Figure 6.4f). With respect to mean f-TP selectivity (hence, also standard deviation), WatDiv covers a lower range of values than DBSB and other benchmarks (cf., Figures 6.4b–6.4c). This is because in DBSB there are unselective queries that return the whole dataset, that is, the subjects, predicates and objects in a triple pattern are all unbound. In contrast, recall that queries were generated in which the predicates in a triple pattern are bound (enabling this feature in WatDiv is a configuration option). Therefore, for this feature WatDiv complements the other benchmarks.

It must be emphasized that WatDiv [18] is not meant to replace existing RDF/SPARQL benchmarks. As noted earlier, WatDiv is only focusing on BGPs, which, for some applications, is not sufficient to cover important test cases. For example, LUBM [91] is an important benchmark for semantic web applications where entailment regimes need to be tested. Likewise, BSBM [44] and SP^{2}Bench [167] cover important use cases such as OPTIONAL and UNION. DBSB [143], on the other hand, is also important, because, first of all, it is based on real queries and second, it is also relatively diverse. The argument that is made by this study is that, WatDiv is more useful in identifying potential issues

with respect to the physical design of systems early on in development (i.e., before systems are deployed and used in production workloads) because WatDiv workloads are more diverse [18], and is suitable for evaluations in this thesis.

6.2 Experiments

In this section, the techniques proposed in this thesis are experimentally evaluated using WatDiv.

6.2.1 Hardware Setup

In the subsequent experiments, a commodity machine is used. The machine is equipped with an AMD Phenom II $\times 4$ 955 3.20 GHz processor, a 16 GB of main memory and a Seagate $3.AA$ hard disk drive with 100 GB of free space at the time of experimentation. The operating system on the machine is Ubuntu 12.04 LTS.

6.2.2 Systems Under Test

RDF data management systems can be classified broadly into two categories with respect to their data representations: (i) tabular and (ii) graph-based. For tabular implementations, one option is to represent data in a single large table. While earlier triplestores followed this approach [51, 55], it has been demonstrated that maintaining redundant copies with different sort orders and indexes can be much more effective [148]. Consequently, the popular prototype RDF-3x [148] (v0.3.7) that follows the latter approach is included in subsequent experiments. It has also been argued that grouping data can significantly improve performance for some workloads [175]. Hence, a second option is to group data by RDF predicates, where data are explicitly partitioned into multiple tables (one table per predicate) and the tables are stored in a column-store [6]. The effectiveness of this approach is tested on MonetDB [112] (v1.7), which is a state-of-the-art column-store. A third option is to natively represent the RDF graph structure, for which the prototype system gStore [200] (v0.2) is used. The tests in this thesis also include three industrial systems, namely, Virtuoso Open Source (VOS) [74] (v6.1.8 and v7.1.0) and 4Store [98] (v1.1.5). Both VOS and 4Store group and index data primarily based on RDF predicates. Furthermore, VOS 6.1 is a row-store and VOS 7.1 is a column-store.

The ideas presented in this thesis are tested using chameleon-db. chameleon-db is implemented in C++, and it consists of more than 35K lines of native source code (excluding the SPARQL parser). Join operations are currently implemented using the hash-join algorithm [70], which has been extended with an adaptation of sideways information passing [147], which is a technique that is also used by x-RDF-3x. The system relies on integer encodings to compress URIs and order-preserving compression to reduce the size of the literals [25]. The dictionary is stored in Berkeley DB [150]. In main memory, each partition is represented as an adjacency list and it is serialized on disk as a consecutive sequence of RDF triples that are sorted on their subject attributes.

6.2.3 Experimental Setup

Unless otherwise stated, the following setup is used in the experiments. For evaluations, the Waterloo SPARQL Diversity Test Suite (WatDiv) is used, because, as argued earlier, it facilitates the generation of test cases that are more diverse than existing benchmarks [18]. In this regard, the WatDiv *data generator* is utilized to create two datasets: one with 10 million RDF triples and another with 100 million RDF triples (it has been observed that systems under test (SUT) load data into main memory on the smaller dataset whereas at 100M triples, SUTs perform disk I/O). Then, using the WatDiv *query template generator*, 125 query templates are created and each query template is instantiated with 100 queries, thus, obtaining 12500 queries in total.

Each system is evaluated independently on each query template. Specifically, for each query template, first, the system is warmed up by executing the workload for that query template once (i.e., 100 queries). Then, the workload is executed five more times (i.e., 500 queries). To reduce and randomize the effects of query interactions, in each run, the queries are shuffled. The average query execution times are reported over the last five workloads. For practical reasons, query execution timeout is set to 60 seconds.

Note that chameleon-db starts with a completely segmented clustering, where each cluster consists of a single triple. For reasons discussed throughout this thesis, this clustering is bad for almost any type of workload: it potentially leads to defragmentation, poor data localization and generation of irrelevant intermediate result tuples. For each query template, chameleon-db is allowed to execute the first 100 queries using this suboptimal clustering, but a timeout threshold is set at 30 minutes (this is in addition to the 60 second query timeout). If the system manages to execute the first 100 queries within 30 minutes, then after the execution of the 100^{th} query, the storage advisor kicks in to compute a better group-by-query clustering. In that case, the last 500 queries are executed over the

group-by-query clustering.[5] This way, with the given timeout threshold, it was possible to collect results for a majority of 92 query templates over the smaller dataset and 76 query templates over the larger one. In the remainder of this chapter, the focus is on only these query templates.

For computing the group-by-query clustering, two techniques are tested independently. The first technique relies on the hierarchical clustering algorithm introduced in Section 4.2 and the second technique is based on TUNABLE-LSH, which was introduced in Section 4.3.

Lastly, note that for chameleon-db, the time to update the underlying physical representation is reflected in the execution times of the first few of the last 500 queries. Furthermore, for the first few queries (of the last 500), indexes are not yet fully constructed and the cache can be cold. In particular, it has been observed that query execution times can improve by an order of magnitude once the indexes are fully constructed and the cache is hot.

6.2.4 Types of Evaluations

In this chapter, three sets of analyses are performed:

- The objective of the first set of analyses is to evaluate the end-to-end performance of query evaluation in chameleon-db, and compare chameleon-db to state-of-the-art RDF data management systems. The idea is to gain insight into when and why chameleon-db performs better than existing systems, as well as when and why it is slower, thus, outlining (and prioritizing) areas of future work. For this set of analyses, chameleon-db is configured to use the clustering techniques introduced in Sections 4.2 and 4.3 (only one at a time), and the trade-offs of using one technique over the other are discussed.

- The objective of the second set of analyses is to understand how (i) physical clustering (cf., Chapters 4.2 and 4.3), (ii) indexing (cf., Section 5.5), and (iii) the query optimization techniques (cf., Chapter 5) proposed in this thesis contribute individually to the overall performance of query evaluation in chameleon-db. For this reason, the experimental logs generated for the above evaluations are sliced and diced across different dimensions.

[5] Improving system performance for the completely segmented clustering or choosing a different initial clustering is beyond the scope of this thesis.

WatDiv 10M Triples	CDB-Hierarchical	CDB-Tunable-LSH	RDF-3x	VOS [6.1]	VOS [7.1]	MonetDB	4Store
Query Execution Time (ms) – geometric mean	**4.7**	19.4	18.8	44.0	24.5	17.0	93.0
% of query templates where system is fastest	**80.4**%	na	1.1%	0.0%	2.2%	16.3%	0.0%
% of query templates where system is fastest	na	**39.0**%	9.8%	0.0%	18.3%	32.9%	0.0%

Table 6.1: End-to-end evaluation of systems on WatDiv 10M triples

WatDiv 100M Triples	CDB-Hierarchical	CDB-Tunable-LSH	RDF-3x	VOS [6.1]	VOS [7.1]	MonetDB	4Store
Query Execution Time (ms) – geometric mean	**40.4**	42.0	71.4	210.3	96.4	62.7	767.2
% of query templates where system is fastest	**48.6**%	na	9.7%	0.0%	0.0%	40.3%	1.4%
% of query templates where system is fastest	na	34.5%	9.1%	0.0%	3.6%	**52.7**%	0.0%

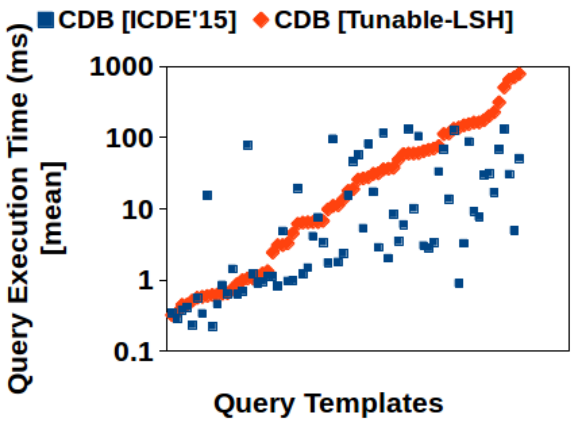
Table 6.2: End-to-end evaluation of systems on WatDiv 100M triples

- Some of the techniques proposed in this thesis, such as TUNABLE-LSH, have a broader applicability than just clustering RDF triples. The objective of the third set of analyses is to evaluate such techniques outside of chameleon-db and in different contexts.

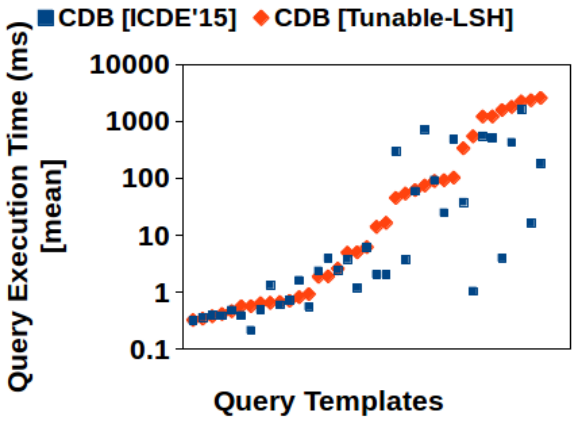
6.2.5 End-to-End Evaluation of Proposed Techniques

In this experiment, two versions of chameleon-db, one in which the G-by-Q clusters are computed using the hierarchical clustering algorithm introduced in Section 4.2 (abbreviated CDB-Hierarchical), and the other in which the G-by-Q clusters are computed using TUNABLE-LSH, which was introduced in Section 4.3, (abbreviated CDB-Tunable-LSH) are compared against five popular RDF data management systems, namely, RDF-3x [148], MonetDB [112], 4Store [98] and Virtuoso Open Source (VOS) versions 6.1 [76] and 7.1 [74].

Tables 6.1 and 6.2 report the mean query execution times (geometric) across all of the query templates discussed in Section 6.2.3, as well as the percentage of query templates for which a particular systems is the fastest. The percentages are reported separately for



(a) WatDiv 10M triples



(b) WatDiv 100M triples

Figure 6.8: Comparison of chameleon-db implemented using a hierarchical clustering algorithm and with Tunable-LSH

the two versions of chameleon-db. Tables 6.1 and 6.2 demonstrate that with the G-by-Q clustering, it is possible to achieve significantly better, consistent performance across a diverse selection of queries than any of the workload-oblivious approaches that have been compared with. That is, for both datasets, at least one version of chameleon-db performs better with the lowest mean query execution time. Furthermore, for the 10M dataset, CDB-Hierarchical is the fastest system for over 80% of the considered 92 query templates.

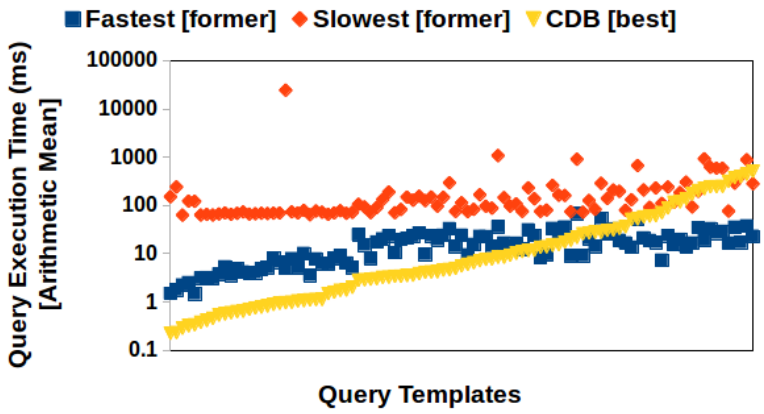
These experiments indicate that on average, the time to compute the G-by-Q clusters

decreases by an order of magnitude with the introduction of Tunable-LSH. For example, for the 100M triples dataset, it takes 317.6 milliseconds on (geometric) average to compute the G-by-Q clusters using the hierarchical clustering algorithm of Section 4.2, whereas with Tunable-LSH, it takes only about 26.1 milliseconds. This is due to the approximate nature of Tunable-LSH. As shown in Tables 6.1 and 6.2 and in Figure 6.8, this approximation has a slight impact on query performance, but for the 100M triples dataset, CDB is still significantly faster than the other RDF data management systems. There is one apparent reason for this: Tunable-LSH is an approximate method, and therefore, the generated G-by-Q clusters are not perfect. To verify this hypothesis, the logs generated during the experiments have been studied further, which revealed the following: using the G-by-Q clustering of Section 4.2 chameleon-db's query engine is able to execute more than 50% of the queries without any decomposition (a property that chameleon-db's query optimizer is trying to achieve [21]), whereas, G-by-Q clustering computed using Tunable-LSH (cf., Section 4.3) has resulted in only 27.1% of the queries to be executed without decomposition. Of course, it is possible to improve chameleon-db's query optimizer further, but that is a topic beyond the scope of this thesis.

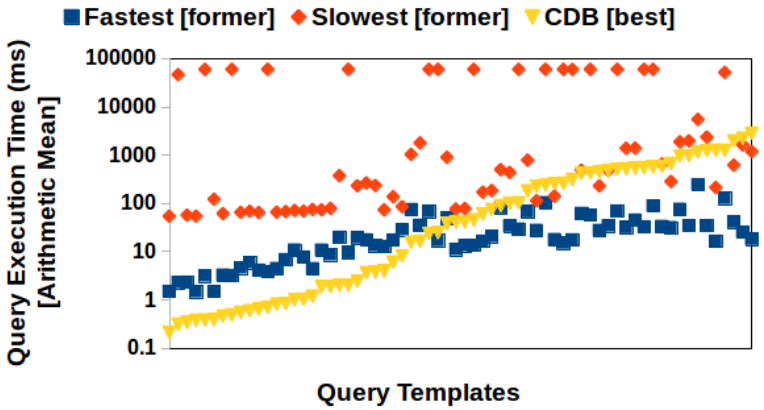
This trade-off between the clustering overhead and the query execution time suggests that for RDF workloads that are too dynamic to be predicted and sampled upfront, it might be desirable to have frequent clustering steps, in which case, using Tunable-LSH is a much better option because of its lower overhead.

Figures 6.9a and 6.9b depict the absolute mean query execution times for each query template. To avoid cluttering the charts, for each query template, the fastest version of chameleon-db is reported. Furthermore, for the remaining systems (i.e., RDF-3x, MonetDB, 4Store, VOS [6.1] and VOS [7.1]), data points for only the fastest and the slowest systems for that particular query template are reported, where the fastest system for one query template is not necessarily the same as that for another query template. It is important to note that for query templates in which chameleon-db is not the fastest system (i.e., the right hand sides of Figures 6.9a and 6.9b), it is still much faster than the slowest system. On the other hand, going from the smaller dataset to the larger, a decrease can be observed in the percentage of queries for which chameleon-db is fastest. Next, this issue is investigated to prioritize potential areas of future work.

First, the improvement in performance due to G-by-Q clustering is quantified to check if there are any anomalies specific to the 100M triples dataset. It is observed that with G-by-Q clustering, there is significant reduction in mean query execution time (details will follow in Section 6.2.6), and the scale of this reduction is consistent in both datasets. Specifically, for 10M triples, query execution becomes faster by a factor of 5.0 and for 100M triples, it becomes faster by a factor of 4.8. This rules out any major anomalies.



(a) For each query template, comparison of chameleon-db against fastest and slowest systems at WatDiv 10M RDF triples



(b) For each query template, comparison of chameleon-db against fastest and slowest systems at WatDiv 100M RDF triples

Figure 6.9: Detailed results

Second, the queries are divided into two groups: (i) those for which chameleon-db is the fastest, and (ii) all the remaining ones. Then, the query logs are analyzed, which keep track of (along with some other information) (i) the query plan the system is using (i.e., OptimalEvaluation vs. SchemalessEvaluation) for a particular query, (ii) as well as the mean G-by-Q cluster size (i.e., in terms of the number of triples) at the time that query was evaluated. Based on this analysis, two important observations can be made (for the

100M triples dataset): (i) in the second group, the mean cluster sizes are about an order of magnitude larger than those in the first one; and (ii) for the first group, 82.9% of the queries have been evaluated with OptimalEvaluation, whereas only 29.7% of queries have been evaluated with OptimalEvaluation in the second group. Upon further manual inspection, it is also noted that in some problematic cases, it would have been possible to choose OptimalEvaluation if stronger but potentially more compute-intensive conditional equivalence rules existed (cf., Chapter 5). Consequently, as future work, better data structures can be developed to improve performance of subgraph matching within each cluster, especially when the clusters are large, Moreover, the query rewriting rules discussed in Chapter 5 can be extended.

Third, cache misses can lead to disproportionately large increases in query evaluation times; therefore, it is important that the system takes full advantage of the G-by-Q clustering. In the prototype implementation of chameleon-db used in these experiments, clusters are serialized in the order they are updated, which can be random due to the locking scheme discussed in Section 4.2.4. More sophisticated serialization techniques to address this problem are left as future work.

Next, a more detailed evaluation is performed by drilling down into particular query features discussed in Section 6.1 (and combinations thereof), which is possible using the WatDiv query analysis tool. Hypothetically speaking, it is possible to perform such analyses using any possible combination of features (including any additional feature not covered by this study). However, the focus in this thesis is on a few special cases where the results stand out, and while doing so, WatDiv's use for stress testing is also demonstrated.

As the first exercise, it is verified whether systems under test including chameleon-db (best of both versions) are biased towards a particular query structure (i.e., linear vs. star/snowflake). To this end, two sets of queries are selected from the test workloads: (i) queries with mean join vertex degree ≤ 3.0 and join vertex count ≥ 3 (representing linear queries), and (ii) queries with mean join vertex degree ≥ 5.0 and join vertex count ≤ 2 (representing star-shaped or snowflake-shaped queries). Multiple conclusions can be drawn from the results in Figure 6.10. First, even though chameleon-db is the fastest system to execute linear queries, it is relatively much slower on linear queries than it is on star or snowflake-shaped queries (hence, room for improvement). Second, the remaining systems are also significantly biased against linear queries. Third, chameleon-db owes its overall improvement in performance to star and/or snowflake-shaped queries because, as Figure 6.10 suggests, it is much faster than other systems for this category of queries.

Next, the systems are tested to determine whether they behave differently for queries in which all (or most) triple patterns contribute almost equally to the overall "selectiveness"

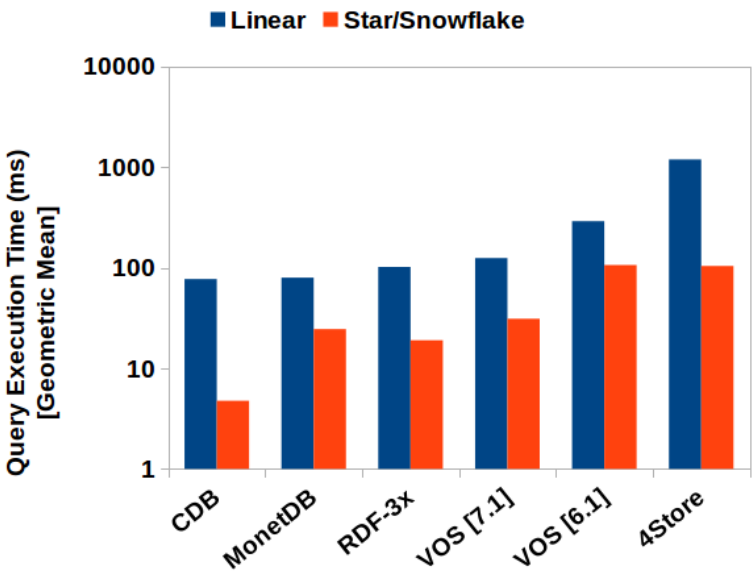


Figure 6.10: Comparison of systems on different query structures using WatDiv 100M

of the query (Case-A) versus the case in which the overall "selectiveness" of the query can be attributed to a single (or few) triple patterns (Case-B). To distinguish between these two cases, the analysis relies on the standard deviation of BGP-restricted f-TP selectivity (cf., Section 6.1), where a low (resp., high) standard deviation implies Case-A (resp., Case-B). For this exercise, only the queries with result cardinality ≤ 500 are taken into account (i.e., selective queries). The spectrum of standard deviation values are divided into three intervals such that there is more or less an equal number of queries in each interval.

Figure 6.11 depicts, for each system, the arithmetic mean of the query execution times of all queries in two of the three intervals. It can be noted that for all systems except chameleon-db, the (mean) query execution times decrease as the standard deviation of BGP-restricted f-TP selectivity increases. This result indicates that, while systems have integrated techniques to early-prune intermediate results [147], these techniques may not be effective for Case-A. For chameleon-db, the opposite is true. An investigation that may reveal a reason for this last observation is left as future work.

The next evaluation reveals how systems scale with increasing result cardinality. For this analysis, the test queries are divided into three groups, namely: (i) query templates with mean result cardinality between $[0, 5)$, (ii) query templates with mean result cardinality between $[5, 50)$, and (iii) query templates with mean result cardinality between

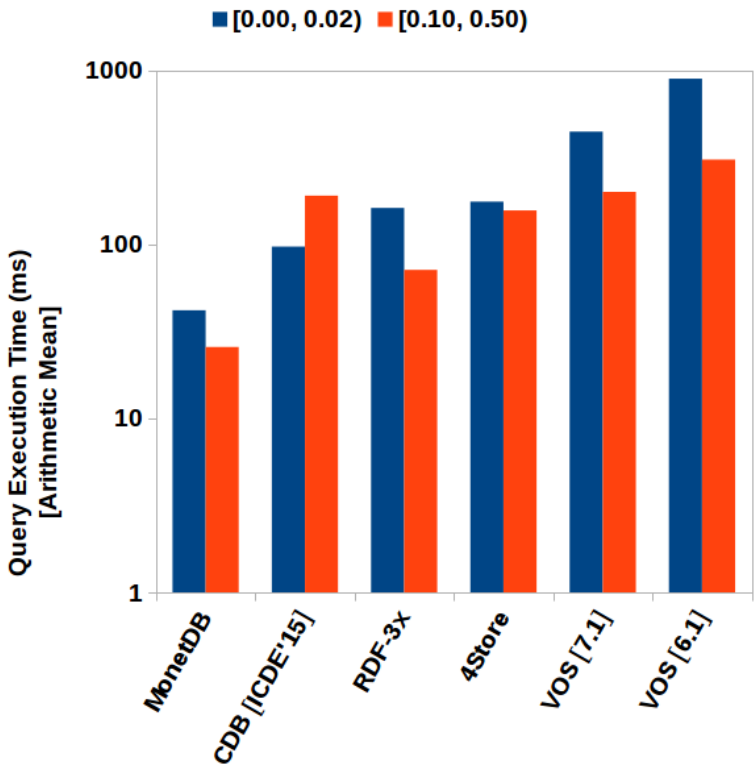


Figure 6.11: Comparison of systems on queries with different BGP-restricted f-TP selectivity (stdev) values using WatDiv 100M

$[50, 500)$.

While the choice of intervals are arbitrary, the experiments demonstrate a strong point. Figure 6.12 suggests that chameleon-db (both versions) does a poor job when it comes to queries with high-cardinality results. There can be multiple reasons for this. First, as suggested earlier, when the result cardinalities are high, chameleon-db is likely accessing multiple G-by-Q clusters, but in its current implementation these clusters are not necessarily physically clustered in the storage system (i.e., inter-cluster serialization). Second, when the result cardinality increases, even a minor deficiency in query optimization can lead to significant drop in performance. Consequently, developing more robust query optimization techniques is an important future work for chameleon-db.

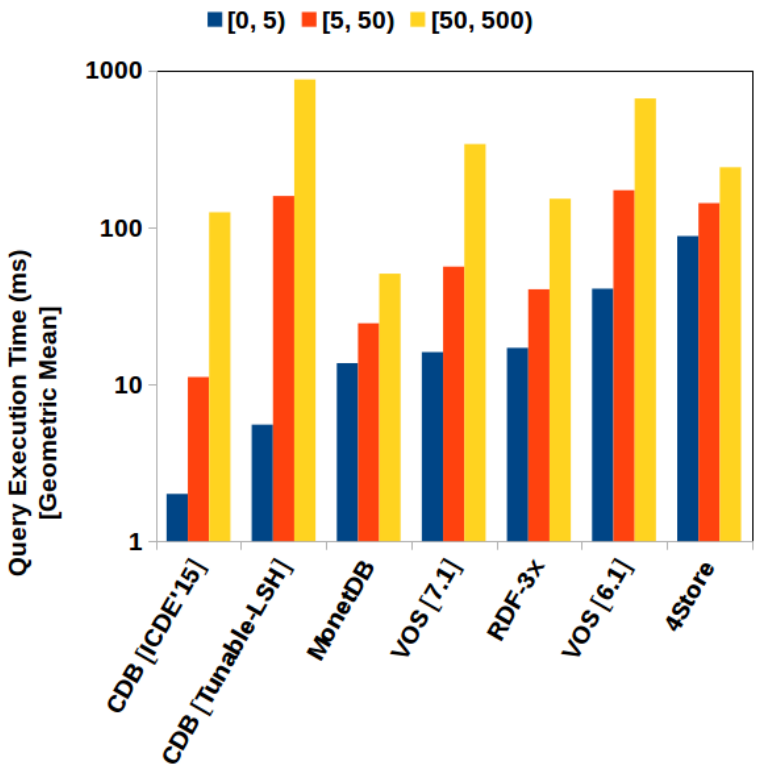


Figure 6.12: Comparison of systems on queries with increasing result cardinality using WatDiv 100M

6.2.6 Evaluation of Individual Components

The purpose of the experiments in this section is to evaluate the individual contribution of the techniques introduced in this thesis to the end-to-end improvements in query execution times. The first analysis quantifies the impact of G-by-Q clustering on query execution. To this end, queries are divided into three categories:

- queries for which the underlying physical clustering is based on a triple-based partitioning of the dataset (i.e., each triple is placed in its own cluster), which corresponds to the first 100 queries in the workload generated for each query template,
- queries for which the underlying G-by-Q clustering is computed using the algorithm discussed in Section 4.2, which corresponds to the latter 500 queries in each workload, and

WatDiv 10M Triples	**Fully Optimized?**				**All**
	Yes		*No*		
Triple-based Clustering	0.55	(3.3%)	70.27	(96.7%)	60.03
Hierarchical Clustering	3.01	(94.9%)	16.00	(5.1%)	3.28
Clustering w/ Tunable-LSH	1.71	(33.8%)	37.48	(66.2%)	13.19

Table 6.3: Evaluation of the impact of different clustering techniques on query execution times using WatDiv 10M triples

WatDiv 100M Triples	**Fully Optimized?**				**All**
	Yes		*No*		
Triple-based Clustering	138.84	(28.8%)	83.40	(71.2%)	96.58
Hierarchical Clustering	5.08	(82.5%)	222.49	(17.5%)	9.84
Clustering w/ Tunable-LSH	1.73	(33.8%)	134.15	(66.2%)	28.18

Table 6.4: Evaluation of the impact of different clustering techniques on query execution times using WatDiv 100M triples

- queries for which the underlying G-by-Q clustering is computed using the algorithm discussed in Section 4.3, which also corresponds to the latter 500 queries in each workload.

To discard (or at least minimize) the effects of indexing on this analysis, queries are filtered out further such that if either the Spill Index or the Cluster Index is updated during the execution of a query, that query is discarded. Furthermore, a distinction is made between queries that cannot be fully optimized (i.e., where the number of query segments is greater than 1) and queries that are fully optimized (i.e., where the number of query segments is equal to 1).

Table 6.3 reports the results over the $10M$ triples dataset, while Table 6.4 reports them over the 100M triples dataset. For each category of queries, the mean (geometric) query execution times (milliseconds) are reported. If applicable, the ratio of queries that are fully optimized to the ratio of queries that are not fully optimized are also reported using percentage values.

Overall, G-by-Q clustering can improve query execution times by an order of magnitude, which makes sense because G-by-Q tries to reduce random I/O and cache stalls, yet, some details are important to mention. First, for the evaluations over the larger dataset and for queries that are fully optimized, the aforementioned statement holds. On the other hand, for the same dataset, the opposite holds if queries are not fully optimized. That is, G-by-Q clustering appears not to make any improvement—in fact, query execution becomes

Query Execution Time, ms (geometric mean)	**WatDiv 10M**	**WatDiv 100M**
Cold Spill Index	15.97	31.01
Hot Spill Index	9.81	22.16

Table 6.5: Impact of cold versus hot Spill Indexes on query execution times

slower for this latter category of queries. This is in-line with the thesis of Chapter 5, which states that *while G-by-Q clustering is a desirable objective, G-by-Q clustering alone cannot guarantee efficient query execution, and techniques are needed for proper query optimization*. As discussed in Chapter 5, in the latter case, irrelevant intermediate results may be generated during query evaluation, which can result in random I/O and cache stalls since G-by-Q clustering does not guarantee that such triples that are irrelevant to the evaluation of the query are physically clustered. Second, for the smaller dataset and for queries that are fully optimized, G-by-Q clustering does not seem to improve query execution times. This is also understandable since the smaller dataset can fit into main memory, where the advantages of G-by-Q clustering over triple-based clustering due to physical clustering may not be visible due to various types of overheads (e.g., locating the correct clusters may become more difficult, subgraph matching within each cluster may become more time-consuming with growing cluster size, etc.) and experimental noise.

In the next set of analyses, the impact of cold versus hot indexes on query execution times are evaluated. Recall from Section 5.5 that indexes in chameleon-db are not built upfront, but as queries are executed, which is similar to the concept of *database cracking* [113]. For these analyses, the Spill Index and the Cluster Index are evaluated separately, and queries are divided into two categories:

- queries during the execution of which, either the Spill Index or the Cluster Index is updated (i.e., cold index), and
- queries during the execution of which, the index in consideration is not updated (i.e., hot index).

Table 6.5 summarizes the results over the Spill Index. These results can be interpreted in two ways. First, the fact that the mean query execution time drops when the index becomes hotter suggests that the indexes are working as desired. That is, as queries are executed, information about the way the RDF graph is partitioned is accumulated in the Spill Index, which is then utilized during query optimization, thus, reducing the total query optimization time. Recall that otherwise, this information needs to be computed

Query Execution Time, ms (geometric mean)	**Triple-based Clustering**	**Hierarchical Clustering**	**Clustering w/ Tunable-LSH**	**All**
Cold Cluster Index	79.14	61.26	75.35	71.99
Hot Cluster Index	31.95	3.86	14.18	9.86

Table 6.6: Evaluation of the impact of cold versus hot Cluster Indexes on query execution times using WatDiv 10M triples

Query Execution Time, ms (geometric mean)	**Triple-based Clustering**	**Hierarchical Clustering**	**Clustering w/ Tunable-LSH**	**All**
Cold Cluster Index	345.94	129.25	392.67	268.29
Hot Cluster Index	48.63	9.49	32.22	21.54

Table 6.7: Evaluation of the impact of cold versus hot Cluster Indexes on query execution times using WatDiv 100M triples

on-the-fly, which can be time consuming. Second, the fact that the mean query execution times even on a cold Spill Index are not unreasonably high suggests that the query rewrite rules introduced in Chapter 5 are also working as desired, because these query rewrite rules have been developed to reduce the amount of information that needs to be collected during query optimization.

Tables 6.6 and 6.7 report the results for the Cluster Index. While some of the arguments made about the Spill Index also apply to the Cluster Index, there is an important difference. The gap between the query execution times over cold versus hot Cluster Indexes is high (i.e., the difference can be an order of magnitude). This suggests that for queries that it sees for the first time, the Cluster Index is not performing as well as desired. This can be improved by building parts of the index upfront, for example, for types of queries that one might predict are common to all (or most) workloads. However, improving the performance of the Cluster Index is beyond the scope of this thesis and is an area of future work. The second point to note is that hot Cluster Indexes work the best for the hierarchical clustering algorithm. Again, this is due to the fact that the underlying clustering becomes fuzzier with Tunable-LSH and even more so with the triple-based partitioning. Therefore, indexes start returning more clusters, likely increasing the number of false-positive clusters that are returned.

Lastly, the impact of the number of join operations on query execution times are evaluated. Note that reducing the number of join operations is one of the primary objectives of the query optimization techniques discussed in Chapter 5. As illustrated in Tables 6.8 and 6.9, as the number of join operations increase, so does the mean query execution time. Consequently, as future work, techniques can be developed such that queries

Join Count	**Hierarchical Clustering**	**Clustering w/ Tunable-LSH**
0	10.66	33.48
[1, 10]	269.43	117.58
[11, 100]]	981.15	220.36
[101, 1000]	5516.19	387.82
[1001, 10000]	N/A	563.95
[10001, 100000]	N/A	6509.60

Table 6.8: Mean query execution time (arithmetic) in milliseconds for query plans with increasing number of join operations on WatDiv 10M triples

Join Count	**Hierarchical Clustering**	**Clustering w/ Tunable-LSH**
0	162.84	25.95
[1, 10]	563.78	412.99
[11, 100]]	501.83	730.82
[101, 1000]	N/A	918.12
[1001, 10000]	N/A	1667.45
[10001, 100000]	N/A	4805.77

Table 6.9: Mean query execution time (arithmetic) in milliseconds for query plans with increasing number of join operations on WatDiv 100M triples

are optimized further. This is especially important for fully utilizing the G-by-Q clustering that is based on Tunable-LSH as it yields less optimal query plans (i.e., ones with more number of join operations) compared to the G-by-Q clustering that is based on the hierarchical clustering algorithm.

6.2.7 Evaluation of Techniques Outside of the Prototype System

Although Tunable-LSH has been developed for computing the contents of G-by-Q clusters, it can be applied to different problems. In this section, Tunable-LSH is evaluated in an in-memory hashtable that has been developed where Tunable-LSH is used to dynamically cluster records in the hashtable. Hashtables are commonly used in RDF data management systems. For example, the dictionary in such a system, which maps integer identifiers to URIs or literals (and vice versa) is often implemented as a hashtable [6,74,188]. Secondary indexes can also be implemented as hashtables, whereby the hashtable acts as a key-value store and maps tuple identifiers to the content of the tuples. In fact, in chameleon-db, all indexes are secondary (dense) indexes.

The hashtable interface is very similar to that of a standard hashtable; except that users are given the option to mark the beginning and end of queries. This information is used to dynamically cluster records such that those that are co-accessed across similar sets of queries also become physically co-located. All of the clustering and re-clustering is transparent to the user, hence, it will be named the *self-clustering hashtable*.

The self-clustering hashtable has the following advantages and disadvantages: Compared to a standard hashtable that tries to avoid hash-collisions, it deliberately co-locates records that are accessed together. If the workloads favour a scenario in which many records are frequently accessed together, then one can expect the self-clustering hashtable to have improved fetch times due to better CPU cache utilization, prefetching, etc. [12]. On the other hand, these optimizations come with three types of overhead. First, every time a query is executed, Tunable-LSH needs to be updated. Second, compared to a standard hashtable in which the physical address of a record is determined solely using the underlying hash function (which is deterministic throughout the entire workload), in this case, the physical address of a record needs to be maintained dynamically because the underlying hash function is not deterministic (i.e., it also changes dynamically throughout the workload). Consequently, there is the overhead of going to a lookup table and retrieving the physical address of a record. Third, physically moving records around in the storage system takes time—in fact, this is often an expensive operation. Therefore, the objectives of this set of experiments are twofold: (i) to evaluate the circumstances

under which the self-clustering hashtable outperforms other popular data structures, and (ii) to understand when the tuning overhead may become a bottleneck. Consequently, the end-to-end query execution times over the self-clustering hashtable are reported, and if necessary, these measurements are broken down into the time to (i) *fetch* the records, and (ii) *tune* the data structures (which includes all types of overhead listed above).

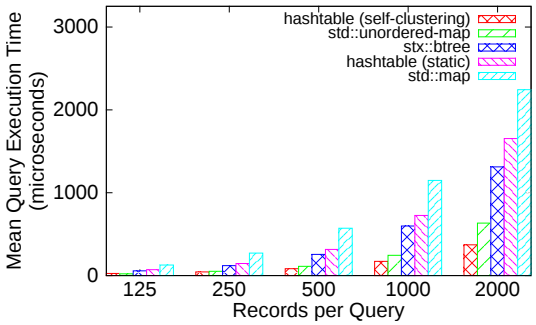
In these experiments, the self-clustering hashtable is compared to popular implementations of three data structures: (i) *std::unordered_map* [3], which is the C++ standard library implementation of a hashtable, (ii) *std::map* [2], which is the C++ standard library implementation of a red-black tree, and (iii) *stx::btree* [40], which is an open source in-memory B+ tree implementation. As a baseline, a static version of the hashtable is also included, i.e., one that does not rely on Tunable-LSH.

The experimental setup in this section is slightly different. More specifically, in the evaluations, two types of workloads are considered: one in which records are accessed *sequentially* and the other in which records are accessed *randomly*. Each workload consists of 3000 queries that are synthetically generated using WatDiv [18]. For each data structure, the end-to-end workload execution times are measured and the mean query execution time is computed by dividing the total workload execution time by the number of queries in the workload.

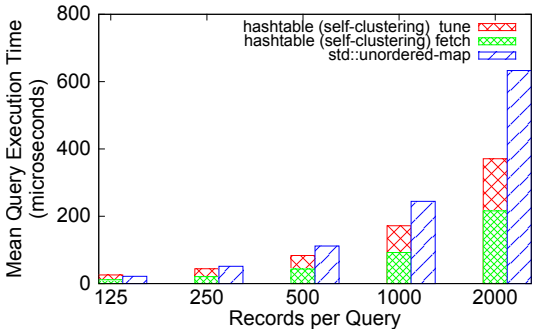
Queries in these workloads consist of changing query access patterns, and in different experiments, different parameters such as the number of records that are accessed by queries on average, the rate at which the query access patterns change in the workload, etc. are controlled. Each experiment is repeated 20 times over workloads that are randomly generated with the same characteristics (e.g., average number of records accessed by each query, how fast the workload changes, etc.) and averages are reported across these 20 runs. Standard errors are not reported because they are negligibly small and they do not add significant value to the results.

For the sequential case, *stx::btree* and *std::map* outperform the hashtables, which is expected because once the first few records are fetched from main-memory, the remaining ones can already be prefetched into the CPU cache (due to the predictability of the sequential access pattern). Therefore, for the remaining part, the focus is on the random access scenario, which is more common in RDF data management systems, and which can be a bottleneck even in systems like RDF-3x [148] that have clustered indexes over all permutations of attributes.

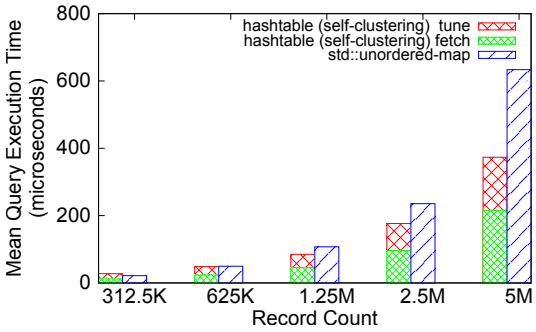
In this experiment, the number of records that a query needs to access (on average) is controlled, where each record is set to 128 bytes. Figure 6.13a compares all the data structures with respect to their end-to-end (mean) query execution times. Three obser-



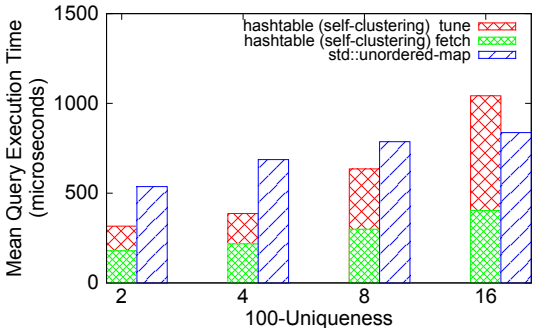
(a) Random Access (All Data Structures) – Control how loaded the workloads are



(b) Random Access – Control how loaded the workloads are



(c) Random Access – Control record count, keep record size constant at 128 bytes



(d) Random Access – Control workload dynamism

Figure 6.13: Experimental evaluation of TUNABLE-LSH in a self-clustering hashtable

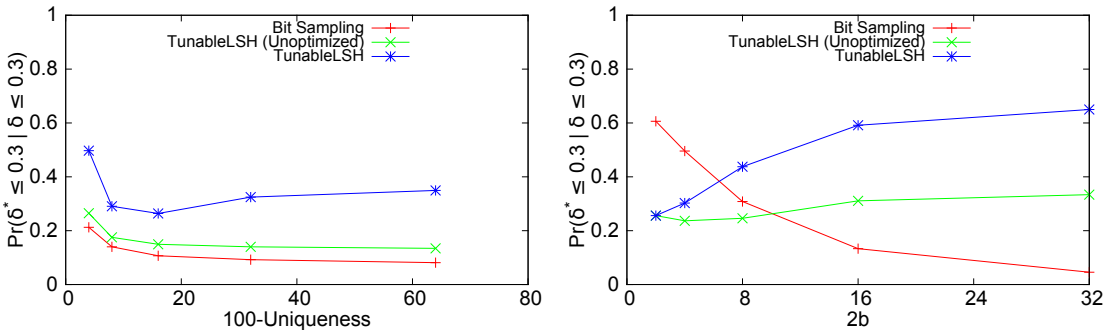
vations stand out: First, in the random access case, the self-clustering hashtable as well as the standard hashtable perform much better than the other data structures, which is what would be expected. This observation holds also for the subsequent experiments, therefore, for presentation purposes, results for these data structures are not included in Figures 6.13b–6.13d. Second, the baseline static version of the hashtable (i.e., without TUNABLE-LSH) performs much worse than the standard hashtable, even worse than a B+ tree. This suggests that my implementation can be optimized further, which might improve the performance of the self-clustering hashtable as well (this is left as future work). Third, as the number of records that a query needs to access increases, the self-clustering hashtable outperforms all the other data structures, which verifies the initial hypothesis.

For the same experiment above, Figure 6.13b focuses on the self-clustering hashtable versus the standard hashtable, and illustrates why the performance improvement is higher (for the self-clustering hashtable) for workloads in which queries access more records. Note that while the *fetch* time of the self-clustering hashtable scales proportionally with respect to std::unordered_map, the *tune* overhead is proportionally much lower for workloads in which queries access more records. This is because with increasing "records per query count", records can be re-located in batches across the pages in main-memory as opposed to moving individual records around.

Next, the average number of records that a query needs to access is kept constant at 2000, but the number of records in the database is controlled. As in the previous experiment, each record is 128 bytes. As illustrated in Figure 6.13c, increasing the number of records in the database (i.e., scaling-up) favours the self-clustering hashtable. The reason is that, when there are only a few records in the database, the records are likely clustered to begin with.

The same experiment is repeated, but this time, by controlling the record size and keeping the database size constant at 640 megabytes. Surprisingly, the relative improvement with respect to the standard hashtable remains more or less constant, which indicates that the improvement is largely dominated by the size of the database, and increasing it is to the advantage of the self-clustering hashtable.

Finally, the sensitivity of the self-clustering hashtable to the dynamism in the workloads is evaluated. Note that for the self-clustering hashtable to be useful at all, the workloads need to be predictable—at least to a certain extent. That is, if records are physically clustered but are never accessed in the future, then all those clustering efforts are wasted. To verify this hypothesis, the expected number of query clusters (i.e., queries with similar but not exactly the same access vectors) in any 100 consecutive queries in the workloads that are generated is controlled. Let this property of the workload be called its 100-*Uniqueness*.



(a) Sensitivity analysis of TUNABLE-LSH– Control workload dynamism

(b) Sensitivity analysis of TUNABLE-LSH– Control TUNABLE-LSH parameter $2b$

Figure 6.14: Experimental evaluation of the sensitivity of TUNABLE-LSH

Figure. 6.13d illustrates how the *tuning* overhead starts to become a bottleneck as the workloads become more and more dynamic, to the extent of being completely unique, i.e., each query accesses a distinct set of records.

In the next set of experiments, the sensitivity of TUNABLE-LSH is evaluated in isolation, that is, without worrying about how it affects physical clustering, and it is compared with three other hash functions: (i) a standard non-locality sensitive hash function [1], (ii) bit-sampling, which is known to be locality-sensitive for Hamming distances [116], and (iii) TUNABLE-LSH without the optimizations discussed in Section 4.3.4. These comparisons are made across workloads with different characteristics (i.e., dense vs. sparse, dynamic vs. stable, etc.) where parameters such as the average number of records accessed per query and the expected number of query clusters within any 100-consecutive sequence of queries in the workload are controlled.

These evaluations indicate that TUNABLE-LSH generally outperforms its alternatives. In the remaining part, the most important observations will be summarized.

Figure 6.14a shows how the probability that *the evaluated hash functions place records with similar utilization vectors to nearby hash values* changes as the workloads become more and more dynamic. In computing these probabilities, both the original distances (i.e., δ) and the distances over the hashed values (i.e., δ^*) are normalized with respect to the maximum distance in each geometry. As illustrated in Figure 6.14a, TUNABLE-LSH achieves higher probability even when the workloads are dynamic. The unoptimized version of TUNABLE-LSH behaves more or less like a static locality-sensitive hash function, such as bit sampling, which is an expected result because TUNABLE-LSH cannot achieve high accuracy without the workload-sensitive arrangement introduced in Section 4.3.4. It

is also important to emphasize that even in that case Tunable-LSH is no worse than a standard LSH scheme, which is aligned with the theorems in Section 4.3.3. The results on the standard non-locality sensitive hash function are not included, because, as one might guess, it has a probability distribution that is completely unparalleled to the clustering objectives of Tunable-LSH.

Figure 6.14b demonstrates how the choice of b (or $2b$ as described in Section 4.3.5) affects the accuracy of Tunable-LSH. Having a higher b implies less and less undesirable collisions of query access vectors, hence, a higher accuracy. On the other hand, for bit sampling, the ideal number of samples is equal to the query clusters in the workload, thus, increasing b, which corresponds to the number of bits that are sampled, might result in oversampling and therefore, lower accuracy. For example, consider two record utilization vectors 1001 and 0001 with Hamming distance 1. If only 1 bit is sampled, there is $\frac{3}{4}$ probability that these two vectors will be hashed to the same value. On the other hand, if 2 bits are sampled, the probability drops to $\frac{1}{2}$.

6.2.8 Discussion

In this section, the techniques proposed in this thesis have been evaluated. These evaluations have aimed to answer the following key questions:

- Can the techniques proposed in this thesis be used to develop systems that are more robust across a diverse selection of SPARQL queries?
- For what types of queries does chameleon-db require further improvements?
- Which components of the prototype system need further improvements?
- Can Tunable-LSH be used in other components of chameleon-db or even in other contexts?

The results in Section 6.2.5 show that chameleon-db is generally more robust than other systems across a diverse selection of SPARQL queries. Nevertheless, chameleon-db needs to be optimized further for (i) queries with high-cardinality results and (ii) linear queries (preferably in this order). The results in Section 6.2.6 verify that to fully exploit the potential optimizations due to G-by-Q clustering, chameleon-db's query optimizer needs to be enhanced. To be more specific, developing and implementing additional query rewrite rules could be one future direction, or implementing a cost-based physical query optimizer could be another (right now, chameleon-db performs only logical query optimization).

Lastly, the results in Section 6.2.7 show that TUNABLE-LSH can be used in contexts other than computing the G-by-Q clusters. For example, it can be used to enhance the lookup speeds in the dictionary of chameleon-db, or it can be used to dynamically determine the inter-cluster serialization order of G-by-Q clusters in the storage system.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis addresses the issue of *building RDF data management systems that can support workloads in which queries are ad-hoc (i.e., diversity) and are frequently changing (i.e., dynamism).* This is non-trivial for many Web applications that have different characteristics than conventional enterprise applications.

Consider conventional enterprise applications such as those used in trade, banking, reservation systems and so on. Each of these applications has a well-defined business logic, which is not very likely to change over time. For example, the schema of a *Customers* relation is not expected to change much—perhaps the most significant change occurred in the last forty years or so when electronic-mails replaced fax, which necessitated the inclusion of an *e-mail* attribute in the schema. In this regard, one can argue that such applications have low tuning requirements from a "database design" point of view. On the other hand, when such changes occur (and they do from time to time), these type of businesses can afford to tune their databases in an *offline* process [135], that is, during scheduled maintenance periods and/or while a back-up server still continues to host the running applications.

In contrast, Web applications have completely different requirements and constraints. On the World Wide Web, things are volatile and highly unpredictable [30, 124, 165]. For example, an event as trivial as Justin Bieber getting himself in trouble with the law can create a chain reaction causing more than a dozen media outlets to cover the story online, thousands of tweets to be generated about the subject, and people reading about these

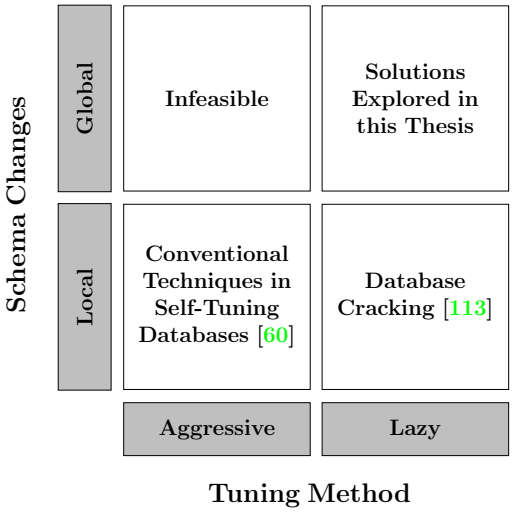


Figure 7.1: Space of solutions in online database tuning

stories/tweets to become curious and issue ad-hoc queries about Justin Bieber—all of this happening within hours, if not minutes. A database management system hosting these queries has only a very short timeframe to tune its physical design to adapt to these changes in the workload (otherwise, after a few hours, users might lose interest in the subject). On the other hand, the tuning overhead to realize these changes can be much higher than anticipated. For example, it might be necessary to switch from a row-oriented representation to a columnar one, all at runtime, which can be hard to achieve using existing techniques [14, 19, 115]. In other words, as applications become more and more ad-hoc and more and more dynamic [30, 124, 165], their tuning overhead increases, while the same does not hold for the tuning budget that is allocated to these applications. This makes Web applications that are ad-hoc and dynamic in nature hard to support using existing database management solutions.

This thesis acknowledges the problem and develops techniques for more adaptive RDF database management systems. The reason RDF data management systems have been chosen as a case study is the fact that RDF is a promising data model for representing graph-structured data (i.e., a significant portion of the data on the Web are graph-structured [35, 42, 43, 119]) and many Web applications have already started utilizing RDF for that purpose [184, 185].

The focus of the thesis has been on developing tuning solutions that are able to make *global* changes in the physical schema of an RDF database in a *lazy* fashion, which is

a problem area that has not been explored in depth before (Figure 7.1). In particular, most of the existing solutions in self-tuning databases [52, 60] work well when the changes made to the physical schema of the database are *local*, e.g., adding indexes for a few more attributes, dropping indexes on a few attributes, horizontally partitioning a table, etc. The main reason for this is that, in this category of solutions, the database is tuned in an *aggressive* fashion. For example, if the tuning advisor in a self-tuning database management system decides to build an index on an attribute, the index is built after all the values of that attribute are fully sorted, which can be a waste of time if the queries do not uniformly look up all these values [115] or if at a later point in time, the tuning advisor decides to drop the index. Database cracking [113–115] improves on this by interleaving index construction with query execution. In other words, given an attribute, its values are *not* fully sorted *upfront*, but instead, these values are partially sorted every time a query is executed [113–115]. This way, it is easier to undo the changes made by each tuning step, should the workloads change. However, database cracking still does not enable any *global* changes to the physical schema of the database. After all, database cracking techniques are implemented for column-stores where each index is built across the values of a column. However, as outlined in Chapter 1, RDF databases are not restricted to column-stores, where different physical designs offer different advantages [19]. Consequently, this thesis takes database cracking a step forward and develops techniques where the physical schema changes can be global. Following are the challenges that have been addressed in this thesis.

- *It is not trivial to determine which physical layout is suitable for which type of SPARQL queries, let alone the fact that experiments demonstrate that no single layout is good for all* (cf., Chapter 1): This challenge has been addressed using a combination of two approaches. First, instead of trying to fit a single layout to the entire database, techniques have been developed that allow different parts of the database to be tuned for different queries in the workload. This way, for example, parts of the database can act as a row-store, while the remaining parts act as a column-store. Second, to determine the best possible physical layout for each of the aforementioned parts of the database, two separate algorithms have been developed (cf., Section 4.2 and Section 4.3). The first algorithm relies on a hierarchical clustering algorithm [21] and the second algorithm relies on a novel locality sensitive hashing scheme that is developed in this thesis called Tunable-LSH [15]. Each algorithm has its advantages and disadvantages. The hierarchical clustering algorithm can more optimally determine the physical layout for each part of the database while taking longer to compute. On the other hand, the algorithm that is based on Tunable-LSH is more approximate, but computationally more efficient.

- *Updating the physical layout can be time consuming*: To address this challenge, the physical layout is updated in a lazy fashion. In other words, the physical layout is updated only for parts of the database that are relevant to the recent queries in the workload (cf., Section 4.2 and Section 4.3). The same principle is used when updating the indexes when the physical layout for parts of the database changes (cf., Section 5.5).

- *When the underlying physical layout is frequently changing, ensuring that queries can be executed correctly and efficiently is not trivial*: First, it must be guaranteed that no matter how the underlying physical layout winds up (as a consequence of executing the algorithms in Sections 4.2 and 4.3), the system has to return correct results to the queries. To this end, Chapter 5 proposes query evaluation techniques that are sound. Second, there is a trade-off between efficiently executing queries and efficiently updating the underlying physical layout. On the one hand, one may choose to index and maintain all information about the underlying physical layout, in which case, query plan generation and execution might be easier, but updating the underlying physical layout becomes harder. On the other hand, one may choose to index and maintain almost nothing about the underlying physical layout, in which case, it is harder to generate query plans, but easier to update the physical layout. To address this second problem, Chapter 5 proposes a solution in-between, where query-rewrite rules are developed that enable efficiently generating query plans while using as little information about the physical layout as possible.

The aforementioned challenges have been addressed and the proposed techniques have been implemented in a prototype system called chameleon-db. The system was solely designed and developed to demonstrate and experiment with the techniques proposed in this thesis. chameleon-db has been implemented in C++ and consists of more than 35 thousand lines of code. Experimental evaluation of these techniques demonstrate that it is possible to build adaptive RDF data management systems. In particular, experimental evaluation of chameleon-db (cf., Chapter 5) demonstrates that (i) the computational overhead of tuning can be contained within less than a second on average (typically on the orders of milliseconds with Tunable-LSH); (ii) chameleon-db can perform as well as most state-of-the-art systems and in some cases even much better than the fastest of the state-of-the-art systems in end-to-end evaluations, which combine the overhead of adaptively updating the underlying physical layout with the time it takes to compute the results of the queries; (iii) chameleon-db is much more robust than state-of-the art systems across a diverse selection of queries; thereby, concluding that the space of solutions explored in this

thesis is a good starting point for dealing with workloads that contain ad-hoc queries that are frequently changing.

7.2 Future Work

While the experimental evaluation of the techniques presented in this thesis are promising, chameleon-db does not yet fully achieve the vision presented in Chapter 1. Following are some of the topics that need further investigation.

7.2.1 Extending the Tuning Model

This thesis does not investigate the problem of "when to tune". In particular, it is assumed that the last k queries are representative of the future queries in a SPARQL workload, where k is set to a constant in most of the experimental evaluation in Chapter 6 (except Section 6.2.7). However, in certain cases, this workload predictability assumption may be too restrictive.

First of all, having a fixed k may not be realistic. Instead, a more realistic scenario is one in which the system dynamically adjusts k based on the current characteristics of the workload. In that case, an important challenge is to balance the cost of tuning and the optimizations achieved as a consequence of tuning.

Second, a preliminary (unpublished) investigation using real SPARQL workloads [38] reveals that different RDF resources may have different fluctuations of popularity. For example, some RDF resources are queried frequently and this frequency does not fluctuate much over time (e.g., over a course of three months). In contrast, for some RDF resources, this fluctuation in frequency can be very high. An initial attempt at modeling these fluctuations using a measure called *volatility* (which is a measure used in economics to model changes in the stock market [81]) has revealed that the volatility of RDF resources follow normal-like distributions [163]. While these results need to be verified further across multiple workloads, they might indicate potential improvements to the proposed techniques (and chameleon-db). For example, for parts of the database that contain RDF resources whose popularity does not fluctuate much over time (i.e., low volatility), a clustering algorithm that is more robust but computationally more intensive such as the one described in Section 4.2 can be used. On the other hand, for parts of the database that contain "volatile" RDF resources, a computationally-efficient but a less accurate clustering algorithm such as the one based on Tunable-LSH in Section 4.3 can be used.

A third challenge that needs to be addressed is dealing with oscillations in the workload. More specifically, there might be cases when the current workload dictates a particular physical layout and at a later point in time, the new workload dictates a different (and potentially contradicting) layout, and this keeps oscillating back and forth. In its current implementation, chameleon-db will keep switching between these two (potentially contradictory) physical layouts, which might be considered a waste of tuning efforts. There can be two solutions to this problem. The first solution is to detect such oscillations in the workload and choose not to tune the database under such circumstances. However a better solution can be to create materialized views for one of the conflicting layouts for such oscillating queries in the workload.

7.2.2 Support for and Optimizations Beyond BGPs

Even though chameleon-db supports SPARQL queries with OPTIONAL graph patterns [157, 159] and UNION [157,159], the system is only optimized for BGPs. Furthermore, the system is not yet well-tested for such complex queries (i.e., there may be bugs). The SPARQL standard [158, 159] is also continuously evolving. For example, now, SPARQL supports *property paths* [158], *regular expressions* [159] and aggregation [158], none of which are supported by chameleon-db. There is also strong incentive towards supporting keyword-search [61,72,181], similarity joins [129] and ranking [73] in RDF data management systems because applications that use RDF typically deal with uncertain and noisy data [127] and RDF datasets might contain a large chunk of unstructured text (i.e., as literal values) that are better located using keyword-search techniques [61, 72, 181]. Consequently, two types of extensions are possible.

The first type of extension is to add new features to chameleon-db that are better aligned with the evolving SPARQL standard and the evolving application requirements. Each of the features mentioned above is a research topic on its own.

The second type of extension is to develop new query optimization strategies for OPTIONAL graph patterns and/or the UNION operation in SPARQL. While some existing techniques try to address these challenges [32, 33], most RDF data management systems rely on optimizations within BGPs.

7.2.3 Improving Existing Query Evaluation and/or Optimization Techniques in chameleon-db for BGPs

The experiments with the clustering algorithm that relies on Tunable-LSH demonstrate that chameleon-db cannot fully optimize queries when the underlying G-by-Q clusters are approximate. This suggests improvements to the existing query evaluation/optimization algorithms in chameleon-db.

A potential improvement is to extend the set of query rewrite rules introduced in Section 5.3. The purpose of these query rewrite rules is to enable efficient generation of query plans even when the system does not know much about the underlying physical layout. As discussed in Chapter 5, this enables the underlying physical layout to be frequently updated. Having a richer set of equivalence rules (i.e., that use more information about the underlying G-by-Q clusters) will enable query plans to be generated for a wider range of G-by-Q clusters such as the ones generated by Tunable-LSH.

A second improvement is to develop and implement a wider range of physical operators. For example, chameleon-db relies only on an implementation of hash join [47, 70, 88] and sideways information passing [147]; however, operators for sort-merge join [70, 88], nested loop join [88], etc. should also be supported in the future. These enhancements will give rise to opportunities for physical query plan optimization, which is largely ignored in the current version of chameleon-db. Furthermore, the implementation of *clustered-match* (cf., Section 5.2) relies on a naïve extension of Ullman's algorithm for subgraph matching [183], which can be extended or improved. In particular, the algorithm does not scale when the individual G-by-Q clusters become too large (cf., Section 6.2.5). Under such circumstances, creating indexes within each G-by-Q cluster or incorporating better query optimization techniques might also help, which is currently omitted in chameleon-db.

A third improvment requires a significant change in the design of chameleon-db. Currently, chameleon-db does not support pipelined execution of queries. Consequently, intermediate results are materialized, which is one of the main reasons why high-cardinality queries are problematic (cf., Section 6.2.5). Furthermore, it is also the reason why some WatDiv stress testing queries time-out when chameleon-db relies on a naïve triple-based partitioning. In particular, it has been observed that in such problematic cases, chameleon-db starts overconsuming main memory resources, thus, triggering paging, which in turn slows down query execution. Supporting a pipelined query execution model [89] might help overcome these problems.

7.2.4 Extending Indexing Techniques

The partial, adaptive indexes used in chameleon-db have the following problems, which need to be improved in the future. First, the index warm-up phase is slow, hence, techniques need to be developed to make it faster. Second, the indexes do not forget what they have learnt from old queries, hence, techniques need to be developed to make the indexes truly adaptive. Third, techniques that make sure that the tree index is balanced needed. Fourth, the workload predictability assumptions mentioned earlier on in this chapter are also applicable to the indexes built in chameleon-db, therefore, they can be exploited to enhance indexing.

7.2.5 Extending Techniques for Adaptively Computing Physical Layouts

In this thesis, techniques are proposed for computing and updating the base physical layout in an RDF data management system. However, there is strong incentive to complement these techniques with view materialization. An important use case is when there are oscillations in the workload: Instead of going back and forth between conflicting physical layouts, materialized views can be created over the relevant parts of the database. The challenge is that this view materialization process should be *online* and *lazy* unlike existing solutions [56, 85, 86], which are *offline* and *aggressive*. Another possible area to explore is whether hybrid techniques that rely on a combination of Tunable-LSH and the hierarchical clustering algorithm presented in this thesis could be employed to better balance the trade-off between robust query execution and easier and faster tuning of the physical layout.

7.2.6 Distributed chameleon-db

This thesis focuses on techniques for scaling-up but not scaling-out. However, the volume of data that RDF data management systems need to deal with is increasing [119], and techniques are needed for the distribution of chameleon-db. An interesting future direction is implementing techniques for the adaptive distribution of RDF, where synergies between existing techniques should be explored [13, 97, 155] (cf., Chapter 3).

7.2.7 Extending WatDiv

Despite its early days, users have started adopting the WatDiv stress testing tools [93, 97, 105]. Based on the feedback obtained from these users, the following items are worth exploring:

- generating dynamic workloads,
- automatically generating dataset description models [16] from existing RDF datasets,
- supporting extensions to SPARQL such as property paths, and
- supporting predicate-predicate joins.

References

[1] std::hash, 2015. http://www.cplusplus.com/reference/functional/hash/.

[2] std::map, 2015. http://www.cplusplus.com/reference/map/map/.

[3] std::unordered_map, 2015. http://www.cplusplus.com/reference/unordered_map/unordered_map/.

[4] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Scalable semantic web data management using vertical partitioning. In *Proc. 33rd Int. Conf. on Very Large Data Bases*, pages 411–422, 2007.

[5] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Using the barton libraries dataset as an RDF benchmark. Technical Report MIT-CSAIL-TR-2007-036, MIT, 2007.

[6] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. SW-Store: a vertically partitioned DBMS for semantic web data management. *VLDB J.*, 18:385–406, 2009.

[7] Daniel J. Abadi, Daniel S. Myers, David J. DeWitt, and Samuel Madden. Materialization strategies in a column-oriented dbms. In *Proc. 23rd Int. Conf. on Data Engineering*, pages 466–475, 2007.

[8] Margareta Ackerman, Shai Ben-David, and David Loker. Towards property-based classification of clustering paradigms. In *Advances in Neural Information Proc. Systems 23, Proc. 24th Annual Conf. on Neural Information Proc. Systems*, pages 10–18, 2010.

[9] Charu C. Aggarwal. A survey of stream clustering algorithms. In *Data Clustering: Algorithms and Applications*, pages 231–258. 2013.

[10] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *Proc. 26th Int. Conf. on Very Large Data Bases*, pages 496–505, 2000.

[11] Sanjay Agrawal, Vivek Narasayya, and Beverly Yang. Integrating vertical and horizontal partitioning into automated physical database design. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 359–370, 2004.

[12] Anastassia Ailamaki, David J. DeWitt, Mark D. Hill, and David A. Wood. DBMSs on a modern processor: Where does time go? In *Proc. 25th Int. Conf. on Very Large Data Bases*, pages 266–277, 1999.

[13] Razen Al-Harbi, Yasser Ebrahim, and Panos Kalnis. PhD-Store: An adaptive SPARQL engine with dynamic partitioning for distributed RDF repositories. *CoRR*, abs/1405.4979, 2014.

[14] Ioannis Alagiannis, Stratos Idreos, and Anastasia Ailamaki. H2O: a hands-free adaptive store. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1103–1114, 2014.

[15] Güneş Aluç, M. Tamer Özsu, and Khuzaima Daudjee. Clustering RDF databases using Tunable-LSH. *CoRR*, abs/1504.02523, 2015.

[16] Güneş Aluç. WatDiv dataset description language tutorial, 2015 (accessed April 22, 2015). http://db.uwaterloo.ca/watdiv/watdiv-schema-tutorial.

[17] Güneş Aluç, David DeHaan, and Ivan T. Bowman. Parametric plan caching using density-based clustering. In *Proc. 28th Int. Conf. on Data Engineering*, pages 402–413, 2012.

[18] Güneş Aluç, Olaf Hartig, M. Tamer Özsu, and Khuzaima Daudjee. Diversified stress testing of RDF data management systems. In *Proc. 13th Int. Semantic Web Conference, Part I*, pages 197–212, 2014.

[19] Güneş Aluç, M. Tamer Özsu, and Khuzaima Daudjee. Workload matters: Why RDF databases need a new design. *Proc. VLDB Endowment*, 7(10):837–840, 2014.

[20] Güneş Aluç, M. Tamer Özsu, Khuzaima Daudjee, and Olaf Hartig. chameleon-db: a workload-aware robust RDF data management system. Technical Report CS-2013-10, University of Waterloo, 2013.

[21] Güneş Aluç, M. Tamer Özsu, Khuzaima Daudjee, and Olaf Hartig. Executing queries over schemaless RDF databases. In *Proc. 31st Int. Conf. on Data Engineering*, pages 807–818, 2015.

[22] Güneş Aluç, M. Tamer Özsu, Khuzaima Daudjee, and Olaf Hartig. Waterloo SPARQL diversity test suite (WatDiv) v0.5, 2015 (accessed June 9, 2015). http://db.uwaterloo.ca/watdiv/.

[23] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proc. 47th Annual Symp. on Foundations of Computer Science*, pages 459–468, 2006.

[24] Renzo Angles, Peter A. Boncz, Josep-Lluis Larriba-Pey, Irini Fundulaki, Thomas Neumann, Orri Erling, Peter Neubauer, Norbert Martínez-Bazan, Venelin Kotsev, and Ioan Toma. The linked data benchmark council: a graph and RDF industry benchmarking effort. *ACM SIGMOD Rec.*, 43(1):27–31, 2014.

[25] Gennady Antoshenkov. Dictionary-based order-preserving string compression. *VLDB J.*, 6(1):26–39, 1997.

[26] Kemafor Anyanwu, HyeongSik Kim, and Padmashree Ravindra. Algebraic optimization for processing graph pattern queries in the cloud. *IEEE Internet Comput.*, 17(2):52–61, 2013.

[27] Marcelo Arenas, Gonzalo I. Diaz, Achille Fokoue, Anastasios Kementsietsidis, and Kavitha Srinivas. A principled approach to bridging the gap between graph data and their schemas. *Proc. VLDB Endowment*, 7(8):601–612, 2014.

[28] Marcelo Arenas, Claudio Gutierrez, and Jorge Pérez. On the semantics of SPARQL. In *Semantic Web Inf. Man.*, pages 281–307. 2009.

[29] Marcelo Arenas and Jorge Pérez. Querying semantic web data with SPARQL. In *Proc. 30th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 305–316, 2011.

[30] Mario Arias, Javier D. Fernández, Miguel A. Martínez-Prieto, and Pablo de la Fuente. An empirical study of real-world SPARQL queries. *CoRR*, abs/1103.5043, 2011.

[31] Vassilis Athitsos, Michalis Potamias, Panagiotis Papapetrou, and George Kollios. Nearest neighbor retrieval using distance-based hashing. In *Proc. 24th Int. Conf. on Data Engineering*, pages 327–336, 2008.

[32] Medha Atre. Optbitmat: For SPARQL OPTIONAL (left-outer-join) queries. *CoRR*, abs/1304.7799, 2013.

[33] Medha Atre. *Left Bit Right*: For SPARQL join queries with OPTIONAL patterns (left-outer-joins). In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1793–1808, 2015.

[34] Medha Atre, Vineet Chaoji, Mohammed J. Zaki, and James A. Hendler. Matrix "bit" loaded: a scalable lightweight join query processor for rdf data. In *Proc. 19th Int. World Wide Web Conf.*, pages 41–50, 2010.

[35] Söen Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, and Zachary Ives. DBpedia: A nucleus for a web of open data. In *Proc. 6th Int. Semantic Web Conference*, pages 11–15, 2007.

[36] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972.

[37] Randall G. Bello, Karl Dias, Alan Downing, James J. Feenan, Jr., James L. Finnerty, William D. Norcott, Harry Sun, Andrew Witkowski, and Mohamed Ziauddin. Materialized views in oracle. In *Proc. 24th Int. Conf. on Very Large Data Bases*, pages 659–664, 1998.

[38] Bettina Berendt, Laura Dragan, Laura Hollink, Markus Luczak-Rösch, Elena Demidova, Stefan Dietze, Julian Szymanski, and John G. Breslin, editors. *Joint Proc. of the 5th International Workshop on Using the Web in the Age of Data and the 2nd International Workshop on Dataset PROFIling and fEderated Search for Linked Data*, volume 1362 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.

[39] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[40] Timo Bingmann. STX B+ tree C++ template classes, 2007. https://panthema.net/2007/stx-btree/.

[41] Bojana Bislimovska, Güneş Aluç, M. Tamer Özsu, and Piero Fraternali. Graph search of software models using multidimensional scaling. In *Proc. of the Workshops of the EDBT/ICDT 2015 Joint Conference*, pages 163–170, 2015.

[42] Chris Bizer. Web of linked data - a global public data space on the Web. In *Proc. 13th Int. Workshop on the World Wide Web and Databases*, 2010.

[43] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[44] Christian Bizer and Andreas Schultz. The Berlin SPARQL benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.

[45] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. of ACM*, 13(7):422–426, 1970.

[46] Mihaela A. Bornea, Julian Dolby, Anastasios Kementsietsidis, Kavitha Srinivas, Patrick Dantressangle, Octavian Udrea, and Bishwaranjan Bhattacharjee. Building an efficient RDF store over a relational database. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 121–132, 2013.

[47] Kjell Bratbergsengen. Hashing methods and relational algebra operations. In *Proc. 10th Int. Conf. on Very Large Data Bases*, pages 323–333, 1984.

[48] Matthias Bröcheler, Andrea Pugliese, and V. S. Subrahmanian. DOGMA: A disk-oriented graph matching algorithm for RDF databases. In *Proc. 8th Int. Semantic Web Conference*, pages 97–113, 2009.

[49] Andrei Z. Broder. On the resemblance and containment of documents. In *Proc. Compression and Complexity of Sequences*, pages 21–29, 1997.

[50] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *J. Comput. Syst. Sci.*, 60(3):630–659, 2000.

[51] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF Schema. In *Proc. 1st Int. Semantic Web Conference*, pages 54–68, 2002.

[52] Nicolas Bruno and Surajit Chaudhuri. To tune or not to tune? a lightweight physical design alerter. In *Proc. 32nd Int. Conf. on Very Large Data Bases*, pages 499–510, 2006.

[53] Min Cai and Martin Frank. RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In *Proc. 13th Int. World Wide Web Conf.*, pages 650–657, 2004.

[54] Jeremy J. Carroll, Christian Bizer, Patrick J. Hayes, and Patrick Stickler. Named graphs, provenance and trust. In *Proc. 14th Int. World Wide Web Conf.*, pages 613–622, 2005.

[55] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: implementing the semantic web recommendations. In *Proc. 13th Int. World Wide Web Conf. - Alternate Track Papers & Posters*, pages 74–83, 2004.

[56] Roger Castillo and Ulf Leser. Selecting materialized views for RDF data. In *Current Trends in Web Engineering - 10th International Conference on Web Engineering, ICWE 2010 Workshops, Vienna, Austria, July 2010, Revised Selected Papers*, pages 126–137, 2010.

[57] Stefano Ceri, Shamkant B. Navathe, and Gio Wiederhold. Distribution design of logical database schemas. *IEEE Trans. Softw. Eng.*, 9(4):487–504, 1983.

[58] Stefano Ceri, Mauro Negri, and Giuseppe Pelagatti. Horizontal data partitioning in database design. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 128–136, 1982.

[59] Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. 34th Annual ACM Symp. on Theory of Computing*, pages 380–388, 2002.

[60] Surajit Chaudhuri and Vivek Narasayya. Self-tuning database systems: a decade of progress. In *Proc. 33rd Int. Conf. on Very Large Data Bases*, pages 3–14, 2007.

[61] Yi Chen, Wei Wang, Ziyang Liu, and Xuemin Lin. Keyword search on structured and semi-structured data. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1005–1010, 2009.

[62] Rada Chirkova and Chen Li. Materializing views with minimal size to answer queries. In *Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 38–48, 2003.

[63] Hyunsik Choi, Jihoon Son, YongHyun Cho, Min Kyoung Sung, and Yon Dohn Chung. Spider: a system for scalable, parallel/distributed evaluation of large-scale RDF data. In *Proc. 18th ACM Int. Conf. on Information and Knowledge Management*, pages 2087–2088, 2009.

[64] Douglas Comer. The ubiquitous b-tree. *ACM Comput. Surv.*, 11(2):121–137, 1979.

[65] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms, 2nd edition*. McGraw-Hill, 2001.

[66] Carlo Curino, Yang Zhang, Evan P. C. Jones, and Samuel Madden. Schism: a workload-driven approach to database replication and partitioning. *Proc. VLDB Endowment*, 3(1):48–57, 2010.

[67] Dean Daniels and Pui Ng. Distributed query compilation and processing in r*. *IEEE Data Eng. Bull.*, 5(3):15–18, 1982.

[68] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. 20th Annual Symp. on Computational Geometry*, pages 253–262, 2004.

[69] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. of ACM*, 51(1):107–113, 2008.

[70] David J. DeWitt, Randy H. Katz, Frank Olken, Leonard D. Shapiro, Michael Stonebraker, and David A. Wood. Implementation techniques for main memory database systems. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1–8, 1984.

[71] Songyun Duan, Anastasios Kementsietsidis, Kavitha Srinivas, and Octavian Udrea. Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 145–156, 2011.

[72] Shady Elbassuoni and Roi Blanco. Keyword search over RDF graphs. In *Proc. 20th ACM Int. Conf. on Information and Knowledge Management*, pages 237–242, 2011.

[73] Shady Elbassuoni, Maya Ramanath, Ralf Schenkel, Marcin Sydow, and Gerhard Weikum. Language-model-based ranking for queries on RDF-graphs. In *Proc. 18th ACM Int. Conf. on Information and Knowledge Management*, pages 977–986, 2009.

[74] Orri Erling. Virtuoso, a hybrid RDBMS/graph column store. *IEEE Data Eng. Bull.*, 35(1):3–8, 2012.

[75] Orri Erling and Ivan Mikhailov. RDF support in the Virtuoso DBMS. In *Proc. 1st Conf. on Social Semantic Web*, pages 59–68, 2007.

[76] Orri Erling and Ivan Mikhailov. RDF support in the Virtuoso DBMS. *Networked Knowledge-Networked Media*, pages 7–24, 2009.

[77] Ronald Fagin, Jürg Nievergelt, Nicholas Pippenger, and H. Raymond Strong. Extendible hashing - A fast access method for dynamic files. *ACM Trans. Database Syst.*, 4(3):315–344, 1979.

[78] Mohammad Farhan Husain, Pankil Doshi, Latifur Khan, and Bhavani Thuraisingham. Storage and retrieval of large rdf graph using hadoop and mapreduce. In *Proc. 1st Int. Conf. Cloud Computing*, pages 680–686, 2009.

[79] Hakan Ferhatosmanoğlu, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. Approximate nearest neighbor searching in multimedia databases. In *Proc. 17th Int. Conf. on Data Engineering*, pages 503–511, 2001.

[80] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice (2nd edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[81] Kenneth R. French, G. William Schwert, and Robert F. Stambaugh. Expected stock returns and volatility. *Journal of Financial Economics*, pages 3–30, 1987.

[82] Luis Galarraga, Katja Hose, and Ralf Schenkel. Partout: a distributed engine for efficient RDF processing. In *Proc. 23rd Int. World Wide Web Conf. – Companion Volume*, pages 267–268, 2014.

[83] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proc. 25th Int. Conf. on Very Large Data Bases*, pages 518–529, 1999.

[84] François Goasdoué, Zoi Kaoudi, Ioana Manolescu, Jorge-Arnulfo Quianè-Ruiz, and Stamatis Zampetakis. CliqueSquare: Flat plans for massively parallel RDF queries. In *Proc. 31st Int. Conf. on Data Engineering*, pages 771–782, 2015.

[85] François Goasdoué, Konstantinos Karanasos, Julien Leblay, and Ioana Manolescu. RDFViewS: a storage tuning wizard for RDF applications. In *Proc. 19th ACM Int. Conf. on Information and Knowledge Management*, pages 1947–1948, 2010.

[86] François Goasdoué, Konstantinos Karanasos, Julien Leblay, and Ioana Manolescu. View selection in semantic web databases. *Proc. VLDB Endowment*, 5(2):97–108, 2011.

[87] Olaf Görlitz, Matthias Thimm, and Steffen Staab. SPLODGE: systematic generation of SPARQL benchmark queries for linked open data. In *Proc. 11th Int. Semantic Web Conference, Part I*, pages 116–132, 2012.

[88] Goetz Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2):73–170, 1993.

[89] Goetz Graefe. Volcano - an extensible and parallel query evaluation system. *IEEE Trans. Knowl. and Data Eng.*, 6(1):120–135, 1994.

[90] Jinghua Groppe, Sven Groppe, Sebastian Ebers, and Volker Linnemann. Efficient processing of SPARQL joins in memory by dynamically restricting triple patterns. In *Proc. 2009 ACM Symp. on Applied Computing*, pages 1231–1238, 2009.

[91] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Semantics*, 3(2-3):158–182, 2005.

[92] Ashish Gupta and Iderpal Singh Mumick, editors. *Materialized views: techniques, implementations, and applications*. MIT Press, Cambridge, MA, USA, 1999.

[93] Sairam Gurajada, Stephan Seufert, Iris Miliaraki, and Martin Theobald. TriAD: a distributed shared-nothing RDF engine based on asynchronous message passing. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 289–300, 2014.

[94] Alon Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[95] Felix Halim, Stratos Idreos, Panagiotis Karras, and Roland H. C. Yap. Stochastic database cracking: Towards robust adaptive indexing main-memory column-stores. *Proc. VLDB Endowment*, 5(6):502–513, 2012.

[96] Richard W. Hamming, editor. *Coding and Information Theory*. Prentice-Hall, Englewood Cliffs, 1986.

[97] Razen Harbi, Ibrahim Abdelaziz, Panos Kalnis, Nikos Mamoulis, Yasser Ebrahim, and Majed Sahli. Adaptive partitioning for very large RDF data. *CoRR*, abs/1505.02728, 2015.

[98] S. Harris, N. Lamb, and N. Shadbolt. 4store: The design and implementation of a clustered RDF store. In *Proc. 5th Int. Workshop on Scalable Semantic Web Knowledge Base Systems*, pages 81–96, 2009.

[99] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. SPARQL 1.1 query language. W3C Recommendation, March 2013.

[100] Andreas Harth and Stefan Decker. Optimized index structures for querying RDF from the Web. In *Proc. of the 3rd Latin American Web Congress*, pages 71–80, 2005.

[101] Andreas Harth, Jürgen Umbrich, Aidan Hogan, and Stefan Decker. Yars2: A federated repository for querying graph structured data from the web. In *Proc. 6th Int. Semantic Web Conference*, pages 211–224, 2007.

[102] Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. Executing SPARQL queries over the web of linked data. In *Proc. 8th Int. Semantic Web Conference*, pages 293–309, 2009.

[103] Olaf Hartig and M. Tamer Özsu. Linked data query processing. In *Proc. 30th Int. Conf. on Data Engineering*, pages 1286–1289, 2014.

[104] Michael Hausenblas. Exploiting linked data to build web applications. *IEEE Internet Comput.*, 13(4):68–73, July 2009.

[105] Joachim Van Herwegen, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. Query execution optimization for clients of triple pattern fragments. In *Proc. 12th European Semantic Web Conference*, pages 302–318, 2015.

[106] Katja Hose and Ralf Schenkel. WARP: Workload-aware replication and partitioning for RDF. In *Proc. 4th Int. Workshop on Data Engineering Meets the Semantic Web*, pages 1–6, 2013.

[107] Katja Hose and Ralf Schenkel. WARP: workload-aware replication and partitioning for RDF. In *Proc. Workshops of the 29th IEEE Int. Conf. on Data Engineering*, pages 1–6, 2013.

[108] Michael E. Houle and Jun Sakuma. Fast approximate similarity search in extremely high-dimensional data sets. In *Proc. 21st Int. Conf. on Data Engineering*, pages 619–630, 2005.

[109] Jiewen Huang, Daniel J. Abadi, and Kun Ren. Scalable SPARQL querying of large RDF graphs. *Proc. VLDB Endowment*, 4(11):1123–1134, 2011.

[110] Jiewen Huang, Kartik Venkatraman, and Daniel J. Abadi. Query optimization of distributed pattern matching. In *Proc. 30th Int. Conf. on Data Engineering*, pages 64–75, 2014.

[111] Mohammad Farhan Husain, James P. McGlothlin, Mohammad M. Masud, Latifur R. Khan, and Bhavani M. Thuraisingham. Heuristics-based query processing for large RDF graphs using cloud computing. *IEEE Trans. Knowl. and Data Eng.*, 23(9):1312–1327, 2011.

[112] Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, K. Sjoerd Mullender, and Martin L. Kersten. MonetDB: Two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull.*, 35(1):40–45, 2012.

[113] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Database cracking. In *Proc. 3rd Biennial Conf. on Innovative Data Systems Research*, pages 68–78, 2007.

[114] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Self-organizing tuple reconstruction in column-stores. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 297–308, 2009.

[115] Stratos Idreos, Stefan Manegold, Harumi A. Kuno, and Goetz Graefe. Merging what's cracked, cracking what's merged: Adaptive indexing in main-memory column-stores. *Proc. VLDB Endowment*, 4(9):585–597, 2011.

[116] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annual ACM Symp. on Theory of Computing*, pages 604–613, 1998.

[117] Paul Jaccard. The distribution of flora in the Alpine zone. *New Phytologist*, 11(2):37–50, 1912.

[118] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31:264–323, 1999.

[119] Anja Jentzsch, Richard Cyganiak, and Chris Bizer. State of the LOD cloud. http://lod-cloud.net/state/, 2011. [Online].

[120] Ruoming Jin, Yang Xiang, Ning Ruan, and David Fuhry. 3-hop: a high-compression indexing scheme for reachability query. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 813–826, 2009.

[121] Zoi Kaoudi and Ioana Manolescu. RDF in the clouds: a survey. *VLDB J.*, 24(1):67–91, 2015.

[122] George Karypis. METIS and parmetis. In *Encyclopedia of Parallel Computing*, pages 1117–1124. 2011.

[123] Jinha Kim, Hyungyu Shin, Wook-Shin Han, Sungpack Hong, and Hassan Chafi. Taming subgraph isomorphism for RDF query processing. *CoRR*, abs/1506.01973, 2015.

[124] Markus Kirchberg, Ryan K. L. Ko, and Bu-Sung Lee. From linked data to relevant data – time is the essence. *CoRR*, abs/1103.5046, 2011.

[125] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and abstract syntax. http://www.w3.org/TR/rdf-concepts/, 2004.

[126] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

[127] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. Test-driven evaluation of linked data quality. In *Proc. 23rd Int. World Wide Web Conf.*, pages 747–758, 2014.

[128] Eugene F. Krause, editor. *Taxicab Geometry: An Adventure in Non-Euclidean Geometry*. Dover, New York, 1986.

[129] Hans-Peter Kriegel, Peter Kunath, Martin Pfeifle, and Matthias Renz. Probabilistic similarity join on uncertain data. In *Proc. 11th Int. Conf. on Database Systems for Advanced Applications*, pages 295–309, 2006.

[130] Joseph B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29:1–27, 1964.

[131] Per-Åke Larson. Linear hashing with overflow-handling by linear probing. *ACM Trans. Database Syst.*, 10(1):75–89, 1985.

[132] Per-Åke Larson. Linear hashing with separators - A dynamic hashing scheme achieving one-access retrieval. *ACM Trans. Database Syst.*, 13(3):366–388, 1988.

[133] Kisung Lee and Ling Liu. Scaling queries over big RDF graphs with semantic hash partitioning. *PVLDB*, 6(14):1894–1905, 2013.

[134] Vladimir I. Levenshtein. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[135] Sam Lightstone, Toby J. Teorey, and Thomas P. Nadeau. *Physical Database Design: the database professional's guide to exploiting indexes, views, storage, and more*. Morgan Kaufmann, 2007.

[136] Witold Litwin. Linear hashing: A new tool for file and table addressing. In *Proc. 6th Int. Conf. on Very Data Bases*, pages 212–223, 1980.

[137] Gang Luo. Partial materialized views. In *Proc. 23rd Int. Conf. on Data Engineering*, pages 756–765, 2007.

[138] Lothar F. Mackert and Guy M. Lohman. R* optimizer validation and performance evaluation for local queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 84–95, 1986.

[139] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[140] Akiyoshi Matono, Said Pahlevi, and Isao Kojima. RDFCube: A P2P-based three-dimensional index for structural joins on distributed triple stores. In *Proc. Databases, Int. Workshops on Information Systems, and Peer-to-Peer Computing*, pages 323–330, 2006.

[141] Knud Möller, Michael Hausenblas, Richard Cyganiak, and Gunnar Aastrand Grimnes. Learning from linked open data usage: Patterns & metrics. In *Proc. of the Web Science Conf.*, pages 1–8, 2010.

[142] Alistair Morrison, Greg Ross, and Matthew Chalmers. Fast multidimensional scaling through sampling, springs and interpolation. *Information Visualization*, 2(1):68–77, 2003.

[143] Mohamed Morsey, Jens Lehmann, Sören Auer, and Axel-Cyrille Ngonga Ngomo. DBpedia SPARQL benchmark - performance assessment with real queries on real data. In *Proc. 10th Int. Semantic Web Conference*, pages 454–469, 2011.

[144] G. M. Morton. A computer oriented geodetic data base; and a new technique in file sequencing. Technical report, IBM Ltd., Ottawa, Canada, 1966.

[145] Fiona Fui-Hoon Nah. A study on tolerable waiting time: how long are Web users willing to wait? *Behaviour & IT*, 23(3):153–163, 2004.

[146] Thomas Neumann and Gerhard Weikum. RDF-3X: a RISC-style engine for RDF. *Proc. VLDB Endowment*, 1(1):647–659, 2008.

[147] Thomas Neumann and Gerhard Weikum. Scalable join processing on very large RDF graphs. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 627–640, 2009.

[148] Thomas Neumann and Gerhard Weikum. The RDF-3X engine for scalable management of RDF data. *VLDB J.*, 19(1):91–113, 2010.

[149] Thomas Neumann and Gerhard Weikum. x-RDF-3X: fast querying, high update rates, and consistency for RDF databases. *Proc. VLDB Endowment*, 3(1):256–263, 2010.

[150] Michael A. Olson, Keith Bostic, and Margo Seltzer. Berkeley DB. pages 43–43, 1999.

[151] Patrick E. O'Neil. Model 204 architecture and performance. In *High Performance Transaction Systems, 2nd International Workshop*, pages 40–59, 1987.

[152] Patrick E. O'Neil and Goetz Graefe. Multi-table joins through bitmapped join indices. *ACM SIGMOD Rec.*, 24(3):8–11, 1995.

[153] Patrick E. O'Neil and Dallan Quass. Improved query performance with variant indexes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 38–49, 1997.

[154] Nikolaos Papailiou, Ioannis Konstantinou, Dimitrios Tsoumakos, and Nectarios Koziris. H2RDF: adaptive query processing on RDF data in the cloud. pages 397–400, 2012.

[155] Nikolaos Papailiou, Dimitrios Tsoumakos, Panagiotis Karras, and Nectarios Koziris. Graph-aware, workload-adaptive SPARQL query caching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1777–1792, 2015.

[156] Peng Peng, Lei Zou, M. Tamer Özsu, Lei Chen, and Dongyan Zhao. Processing SPARQL queries over linked data-a distributed graph-based approach. *CoRR*, abs/1411.6763, 2014.

[157] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):1–45, 2009.

[158] Eric Prud'hommeaux, Steve Harris, and Andy Seaborne. SPARQL 1.1 Query Language. http://www.w3.org/TR/sparql11-query, 2013.

[159] Eric Prudhommeaux and Andy Seaborne. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 2008.

[160] Martin Przyjaciel-Zablocki, Alexander Schätzle, Eduard Skaley, Thomas Hornung, and Georg Lausen. Map-side merge joins for scalable SPARQL BGP processing. In *Proc. 5th Int. Conf. Cloud Computing*, pages 631–638, 2013.

[161] Shi Qiao and Z. Meral Özsoyoglu. Rbench: Application-specific RDF benchmarking. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1825–1838, 2015.

[162] Padmashree Ravindra and Kemafor Anyanwu. Scaling unbound-property queries on big RDF data warehouses using mapreduce. In *Advances in Database Technology, Proc. 18th Int. Conf. on Extending Database Technology*, pages 169–180, 2015.

[163] William Reed. The normal-Laplace distribution and its relatives. In *Proc. Advances in Distribution Theory, Order Statistics, and Inference*, pages 61–74, 2006.

[164] Kurt Rohloff and Richard E. Schantz. High-performance, massively scalable distributed systems using the MapReduce software framework: the SHARD triple-store. In *SPLASH Workshop on Programming Support Innovations for Emerging Distributed Applications*, page 4, 2010.

[165] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proc. 19th Int. World Wide Web Conf.*, pages 851–860, 2010.

[166] Alexander Schätzle, Martin Przyjaciel-Zablocki, Antony Neu, and Georg Lausen. Sempala: Interactive SPARQL query processing on hadoop. In *Proc. 13th Int. Semantic Web Conference, Part I*, pages 164–179, 2014.

[167] Michael Schmidt, Thomas Hornung, Michael Meier, Christoph Pinkel, and Georg Lausen. Sp^2bench: A sparql performance benchmark. In *Semantic Web Inf. Man.*, pages 371–393. 2009.

[168] Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of SPARQL query optimization. In *Proc. 13th Int. Conf. on Database Theory*, pages 4–33, 2010.

[169] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proc. 1st IEEE Int. Conf. on Peer-to-Peer Computing*, pages 101–102, 2001.

[170] Guus Schreiber and Yves Raimond. RDF 1.1 primer. W3C Note, February 2014.

[171] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. FedX: Optimization techniques for federated query processing on linked data. In *Proc. 10th Int. Semantic Web Conference*, pages 601–616, 2011.

[172] Praveen Seshadri, Joseph M. Hellerstein, Hamid Pirahesh, T. Y. Cliff Leung, Raghu Ramakrishnan, Divesh Srivastava, Peter J. Stuckey, and S. Sudarshan. Cost-based optimization for magic: Algebra and implementation. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 435–446, 1996.

[173] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[174] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Proc. IEEE 26th Symp. on Mass Storage Systems and Technologies*, pages 1–10, 2010.

[175] Lefteris Sidirourgos, Romulo Goncalves, Martin Kersten, Niels Nes, and Stefan Manegold. Column-store support for RDF data management: not all swans are white. *Proc. VLDB Endowment*, 1(2):1553–1563, 2008.

[176] Markus Stocker, Andy Seaborne, Abraham Bernstein, Christoph Kiefer, and Dave Reynolds. Sparql basic graph pattern optimization using selectivity estimation. In *Proc. 17th Int. World Wide Web Conf.*, pages 595–604, 2008.

[177] Michael Stonebraker. The case for shared nothing. *IEEE Data Eng. Bull.*, 9(1):4–9, 1986.

[178] Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alex Rasin, Nga Tran, and Stanley B. Zdonik. C-store: A column-oriented DBMS. In *Proc. 31st Int. Conf. on Very Large Data Bases*, pages 553–564, 2005.

[179] Wen Sun, Achille Fokoue, Kavitha Srinivas, Anastasios Kementsietsidis, Gang Hu, and Guotong Xie. SQLGraph: An efficient relational-based property graph store. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1887–1901.

[180] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. *ACM Trans. Database Syst.*, 35(3):20:1–20:46, 2010.

[181] Thanh Tran, Haofen Wang, Sebastian Rudolph, and Philipp Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In *Proc. 25th Int. Conf. on Data Engineering*, pages 405–416, 2009.

[182] Octavian Udrea, Andrea Pugliese, and V. S. Subrahmanian. Grin: a graph based RDF index. In *Proc. 22nd National Conf. on Artificial Intelligence*, pages 1465–1470, 2007.

[183] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

[184] Ruben Verborgh, Olaf Hartig, Ben De Meester, Gerald Haesendonck, Laurens De Vocht, Miel Vander Sande, Richard Cyganiak, Pieter Colpaert, Erik Mannens, and Rik Van de Walle. Querying datasets on the web with high availability. In *Proc. 13th Int. Semantic Web Conference, Part I*, pages 180–196, 2014.

[185] Ruben Verborgh, Erik Mannens, and Rik Van de Walle. Initial usage analysis of DBpedia's triple pattern fragments. In *Joint Proc. of the 5th International Workshop on Using the Web in the Age of Data and the 2nd International Workshop on Dataset PROFIling and fEderated Search for Linked Data*, pages 1–11, 2015.

[186] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. Hexastore: sextuple indexing for semantic web data management. *Proc. VLDB Endowment*, 1(1):1008–1019, 2008.

[187] Tom White. *Hadoop - The Definitive Guide: MapReduce for the Cloud*. O'Reilly, 2009.

[188] Kevin Wilkinson. Jena property table implementation. Technical Report HPL-2006-140, HP-Labs, 2006.

[189] Kevin Wilkinson, Craig Sayers, Harumi Kuno, and Dave Reynolds. Efficient RDF storage and retrieval in Jena2. In *Proc. 1st Int. Workshop on Semantic Web and Databases*, pages 131–150, 2003.

[190] Ying Yan, Chen Wang, Aoying Zhou, Weining Qian, Li Ma, and Yue Pan. Efficient indices using graph partitioning in RDF triple stores. In *Proc. 25th Int. Conf. on Data Engineering*, pages 1263–1266, 2009.

[191] Hilmi Yildirim, Vineet Chaoji, and Mohammed J. Zaki. Grail: scalable reachability index for large graphs. *Proc. VLDB Endowment*, 3(1):276–284, 2010.

[192] Pingpeng Yuan, Pu Liu, Buwen Wu, Hai Jin, Wenya Zhang, and Ling Liu. TripleBit: a fast and compact system for large scale RDF data. *Proc. VLDB Endowment*, 6(7):517–528, 2013.

[193] Kai Zeng, Jiacheng Yang, Haixun Wang, Bin Shao, and Zhongyuan Wang. A distributed graph engine for web scale RDF data. *Proc. VLDB Endowment*, 6(4):265–276, 2013.

[194] Shijie Zhang, Jiong Yang, and Wei Jin. Sapper: subgraph indexing and approximate matching in large graphs. *Proc. VLDB Endowment*, 3(1):1185–1194, 2010.

[195] Peixiang Zhao and Jiawei Han. On graph query optimization in large networks. *Proc. VLDB Endowment*, 3(1):340–351, 2010.

[196] Peixiang Zhao, Jeffrey Xu Yu, and Philip S. Yu. Graph indexing: tree + delta <= graph. In *Proc. 33rd Int. Conf. on Very Large Data Bases*, pages 938–949, 2007.

[197] Jingren Zhou, Per-Åke Larson, Jonathan Goldstein, and Luping Ding. Dynamic materialized views. In *Proc. 23rd Int. Conf. on Data Engineering*, pages 526–535, 2007.

[198] Daniel C. Zilio, Jun Rao, Sam Lightstone, Guy M. Lohman, Adam J. Storm, Christian Garcia-Arellano, and Scott Fadden. DB2 design advisor: Integrated automatic physical database design. In *Proc. 30th Int. Conf. on Very Large Data Bases*, pages 1087–1097, 2004.

[199] Lei Zou, Lei Chen, Jeffrey Xu Yu, and Yansheng Lu. A novel spectral coding in a large graph database. In *Advances in Database Technology, Proc. 11th Int. Conf. on Extending Database Technology*, pages 181–192, 2008.

[200] Lei Zou, Jinhui Mo, Dongyan Zhao, Lei Chen, and M. Tamer Özsu. gStore: Answering SPARQL queries via subgraph matching. *Proc. VLDB Endowment*, 4(1):482–493, 2011.

[201] Lei Zou, M. Tamer Özsu, Lei Chen, Xuchuan Shen, Ruizhe Huang, and Dongyan Zhao. gStore: a graph-based SPARQL query engine. *VLDB J.*, 23(4):565–590, 2014.