

Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften (Dr.-Ing.)
von der Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)
genehmigte Dissertation von
Dipl.-Inform.Wirt. Andreas Josef Wagner

RANK-AWARE, APPROXIMATE QUERY PROCESSING ON THE SEMANTIC WEB

ANDREAS JOSEF WAGNER

Tag der mündlichen Prüfung: 11.06.2014

Referent: Prof. Dr. Rudi Studer

Korreferent: Prof. Dr. Wolfgang Nejdl

Karlsruhe, 2014

*This thesis is dedicated to my loving family – my mother and father,
brother, sister-in-law, and niece.*

ABSTRACT

The amount of data on the World Wide Web that adheres to Semantic Web standards is rapidly increasing. Most notably, many Web pages are annotated with RDFa, Microdata, or Microformats. Moreover, the popular Linked Data principles led to a drastic increase in Semantic Web data, which is accessible using simple HTTP operations. In fact, not only instance data, but also schema data is published on the Web. Thus, one may conceive the World Wide Web as a vast space of interlinked data sources, which feature Semantic Web data.

Search over this huge Web data corpus frequently leads to queries having large result sets. So, in order to discover data elements, which satisfy a given information need, users must rely on ranking techniques to sort results according to their relevance. Unfortunately, processing queries with ranked results over a large data corpus is highly expensive in terms of computation time as well as computation resources. This is because *the sorting of query results is a blocking operation in the query operator tree*. In simple terms, all results have to be computed, before they can be sorted according to their assigned ranking scores. Clearly, processing queries in this manner causes prohibitive query computation costs – in particular, in context with web-scale data.

At the same time, applications oftentimes face information needs, which do not require complete and exact results. Most notably, end-users who search the Web commonly only investigate a small fraction of top-ranked query results, until they discover a data element of interest. Thus, *applications should be able to process queries in a flexible way – some queries may require exact results, while others could be answered approximately*.

In this thesis, we face the problem of *how to process queries over Web data in an approximate and rank-aware fashion*. Aiming at this complex problem, we provide several novel contributions.

More specifically, we introduce a *rank-aware join operator for Web data*. By means of this join operator, we can process queries with ranked results much more efficiently. That is, our rank-aware join operator focuses on computing the top-ranked query results first, while omitting the remainder of the results. This way, these join operators consume much less join inputs, which translates to performance gains for the overall query computation time.

Additionally, we exploit the fact that many information needs can be addressed via incomplete/approximated result sets. That is, we enable systems to trade-off result completeness and accuracy, in favor of query computation time. We provide two contributions for this *approximate query processing*. On the one hand, we present a novel *pipeline of operations, which allows to incrementally compute query results*. So, initial approximate results can either be reported directly or could be refined as needed. On the other hand, we introduce a new *approximate rank-aware join operator*. Our operator allows to discard such intermediate query results, which are not likely to lead to a final top-ranked result. In other

words, the approximate rank-aware join operator enables a system to discard low-ranked results during the query processing.

Furthermore, we present a novel approach for selectivity estimation that is tailored towards the needs of Web data and typical Web queries. That is, our selectivity estimation approach allows the estimation of queries, which match structured as well as unstructured data elements in the Web of data. Such a selectivity estimation is crucial for query optimization techniques, which can integrate our approximate/rank-aware join operators in physical query plans.

PUBLICATIONS

Text as well as figures in this thesis have partly been already published. That is, the thesis is based on the following papers:

- [1] Tran, Thanh and Ladwig, Günter and Wagner, Andreas, **Approximate and Incremental Processing of Complex Queries against the Web of Data**. Proceedings of the Database and Expert Systems Applications Conference (DEXA), 2011.
- [2] Wagner, Andreas and Duc, Thanh Tran and Ladwig, Günter and Harth, Andreas and Studer, Rudi, **Top-k Linked Data query processing**. Proceedings of the Extended Semantic Web Conference (ESWC), 2012.
- [3] Wagner, Andreas and Bicer, Veli and Tran, Thanh, **Selectivity Estimation for Hybrid Queries over Text-Rich Data Graphs**. Proceedings of the International Conference on Extending Database Technology (EDBT), 2013.
- [4] Wagner, Andreas and Bicer, Veli and Tran, Thanh, **Pay-as-you-go Approximative Top-k Join Processing for the Web of Data**. Proceedings of the Extended Semantic Web Conference (ESWC), 2014.

CONTENTS

1	INTRODUCTION	2
1.1	Motivation	2
1.1.1	Semantic Web Data	2
1.1.2	Semantic Search	6
1.2	Web Data Characteristics	10
1.2.1	Schemaless Data	10
1.2.2	Hybrid Data	10
1.2.3	Distributed and Low-volume Data	11
1.3	Research Questions and Scope	11
1.3.1	Research Questions	11
1.3.2	Scope of this Thesis	13
1.4	Contributions	13
1.5	Outline	15
2	FOUNDATIONS	17
2.1	Data and Query Model	17
2.1.1	Structured and Unstructured Data	17
2.1.2	Structured and Unstructured Queries	20
2.2	Query Processing	23
2.2.1	Overview	23
2.2.2	Query Optimization	25
2.2.3	Cost Model	28
2.2.4	Query Execution	29
2.3	Rank-aware Query Processing	30
2.3.1	Overview	31
2.3.2	Top-k Join Processing	32
3	RANK-AWARE QUERY PROCESSING	38
3.1	Introduction	40
3.1.1	Motivation	40
3.1.2	Data-driven Linked Data Query Processing	42
3.1.3	Problem	45
3.2	Research Questions and Contributions	46
3.2.1	Research Questions and Hypotheses	46
3.2.2	Contributions	47
3.3	Linked Data Top-k Query Processing	48
3.3.1	Sorted Access	48
3.3.2	Push-based Top-k Join Processing	50
3.3.3	Improved Threshold Estimation	54
3.3.4	Early Pruning of Partial Results	58
3.4	Evaluation	59
3.4.1	Evaluation Setting	60

3.4.2	Evaluation Results	61
3.5	Related Work	64
3.5.1	Pull-based, Centralized Top-k Processing	64
3.5.2	Distributed Top-k Processing	64
3.5.3	Approximate Top-k Processing	65
3.6	Summary	65
4	SELECTIVITY ESTIMATION	68
4.1	Introduction	69
4.1.1	Motivation	69
4.1.2	Selectivity Estimation	71
4.1.3	Probabilistic Framework	72
4.1.4	Problem	76
4.2	Research Questions and Contributions	77
4.2.1	Research Questions and Hypotheses	77
4.2.2	Contributions	78
4.3	Selectivity Estimation over Text-Rich RDF Graphs	79
4.3.1	Data Synopsis	79
4.3.2	Data Synopsis Construction	84
4.3.3	Selectivity Estimation	89
4.4	Evaluation	92
4.4.1	Evaluation Setting	93
4.4.2	Evaluation Results: Effectiveness	97
4.4.3	Evaluation Results: Efficiency	98
4.5	Related Work	99
4.6	Summary	100
5	APPROXIMATE QUERY PROCESSING	103
5.1	Motivation	105
5.2	Approximate Incremental Query Processing	107
5.2.1	Introduction	107
5.2.2	Research Questions and Contributions	108
5.2.3	A Pipeline-based Approach for Approximate and Incremental Query Processing	109
5.2.4	Evaluation	123
5.2.5	Related Work	128
5.2.6	Summary	129
5.3	Rank-aware Approximate Query Processing	131
5.3.1	Introduction	131
5.3.2	Research Questions and Contributions	134
5.3.3	Pay-as-you-go Approximate Top-k Join Processing	135
5.3.4	Evaluation	153
5.3.5	Related Work	159
5.3.6	Summary	160
5.4	Conclusion	161
6	CONCLUSION	164
6.1	Summary	164

6.2 Future Work	167
BIBLIOGRAPHY	169
List of Figures	184
List of Tables	186
List of Algorithms	187
Acronyms	188
A APPENDIX: EVALUATION QUERIES	189
A.1 Evaluation Queries for Chapter 3	190
A.2 Evaluation Queries for Chapter 4	197
A.3 Evaluation Queries for Chapter 5	213

INTRODUCTION

INTRODUCTION

Context of this Thesis. In this thesis, we are concerned with efficiency aspects of search on Web data. While these efficiency problems are manifold in nature, we specifically target ranking and approximation aspects.

Our research led to the development of an open-source Web data management system (CumulusRDF), which is freely available on the Web.¹ Amongst other deployments, CumulusRDF is currently used in the iZEUS² research project, where the system manages real-time data from electric vehicles.

Running Example. Throughout the thesis, we follow a running example about the movie “Roman Holiday”. For instance, Figure 1 illustrates the IMDB Web page about “Roman Holiday”. In particular, we will explain our approaches and contributions by means of this example. Note, while this example is situated in the movie domain, our approaches are generic and can be used on data from any domain.

1.1 MOTIVATION

1.1.1 Semantic Web Data

The terminology *Web data* and *Web search* refers to data and search on the World Wide Web (www). The amount of Web documents is large and rapidly increasing. As of October 2013, more than 767 million Web pages are available on the WWW.³ Traditionally, data published on the WWW is represented as simple HyperText Markup Language ([HTML](http://www)) pages, with no means for machines to “understand” data semantics. That is, HTML pages solely specify how machines, e.g., Web browsers, are supposed to visualize the data, but not *what the pages are about and how they relate to each other* [81].

Example 1

Consider the HTML page about the movie “Roman Holiday” in Figure 1. Only a human reader understands that this page describes a famous movie called “Roman Holiday” featuring Audrey Hepburn. However, a machine

¹<http://code.google.com/p/cumulusrdf/>, retrieved 2013-10-05.

²<http://www.izeus.kit.edu/>, retrieved 2013-10-10.

³<http://news.netcraft.com/>, retrieved 2013-10-05.



Figure 1: HTML IMDB Web page about the movie “Roman Holiday” visualized by a browser.⁴

only sees the HTML data, which dictates how to visualize the contents of that page, see Figure 2.

Addressing this problem, the World Wide Web Consortium (W₃C) proposed several open standards (commonly referred to as *Semantic Web Stack*) for encoding Web data such that machines are able to infer meaning from that data – thereby forming the *Semantic Web*.⁵

More precisely, the W3C proposed Uniform Resource Identifiers (URIs)⁶ as entity identifiers and defined the Resource Description Format (RDF) [147] and the Resource Description Framework Schema (RDFS) [29] as lightweight Web data formats. Further, the SPARQL protocol and RDF query language (SPARQL) was proposed as query language for RDF data [17]. The interested reader may find a detailed introduction to the complete Semantic Web Stack in [81].

For the remainder of this thesis, we define Semantic Web data, also referred to as *Web data*, as follows [160]:

⁴<http://www.imdb.com/title/tt0046250/>, retrieved 2013-10-05.

⁵<http://www.w3.org/standards/semanticweb/>, retrieved 2013-10-05.

⁶<http://tools.ietf.org/html/rfc3986>, retrieved 2013-10-07.

```

634 <div class="image">
635   <a href="/media/rm1442547968/tt0046250?ref_tt_ov_1">
636     
641   </a>
642 </div>
643 </td>
644 <td id="overview-top">
645
646   <div id="prometer_container">
647     <div id="prometer" class="meter-collapsed down">
648       <div id="meterHeaderBox">
649         <div id="meterTitle" class="meterToggleOnHover">MOVIEmeter</div>
650         <a onClick="(new Image()).src='/rg/tt_moviemeter_why/prosystem/images/b.gif?link=/r/tt_moviemeter_why/ttle/tt0046250';"
651           href="/r/tt_moviemeter_why/ttle/tt0046250"
652           id="meterRank">Top 5000</a>
653       </div>
654       <div id="meterChangeRow" class="meterToggleOnHover">
655         <span>Down</span>
656         <span id="meterChange">351</span>
657         <span>this week</span>
658       </div>
659       <div id="meterSeeMoreRow" class="meterToggleOnHover">
660         <a onClick="(new Image()).src='/rg/tt_moviemeter_why/prosystem/images/b.gif?link=/r/tt_moviemeter_why/ttle/tt0046250';"
661           href="/r/tt_moviemeter_why/ttle/tt0046250">View rank on IMDbPro</a>
662       <span>&raquo;</span></div>
663     </div>
664   </div>
665 </div>
666 <h1 class="header"> Roman Holiday
667   <span class="nobr"><a href="/year/1953/?ref_tt_ov_inf" >1953</a></span>
668 </h1>
669 <div class="infobar">
670   <span title="Not Rated"
671     class="us_not_rated titlePageSprite absmiddle"
672     content="NOT RATED"></span>

```

Figure 2: HTML source code for the “Roman Holiday” Web page in Figure 1. This structured HTML data only specifies how to display the data.

»» Definition 1: Semantic Web Data, Web Data (Informal) [160]

Semantic Web data (Web data) contains descriptions of entities on the Web, with each description being a set of triples: $\{(s, p, o)\}$. A triple associates an entity (subject) s with an object o via a predicate p . A set of triples forms a data graph.

Note, we will provide a formal data model in Chapter 2.

In recent years, the amount of Web data drastically increased. Most notably, semantic annotations, Linked Data,^{7,8} and Web schemata have contributed to this development [160].

- ① Semantic annotations are structured data elements, which can be embedded in Web pages. For this, RDFa [11], Microdata [80], and Microformats⁹ have been used. For estimating the amount of such annotations, the Web Data Commons project¹⁰ recently analyzed the Common Crawl¹¹ – a well-known corpus of 3 billion unique HTML pages retrieved from 40.6 million domains. According to [25] and the Web Data Commons project, 12.3% of the websites contained in the Common Crawl corpus contain structured data. In terms of RDF, this structured data is represented by means of 7.3 billion RDF triples and captures 1.15 billion typed entities. Other studies even estimate that approximately 10% of all Web pages feature semantic annotations [121].

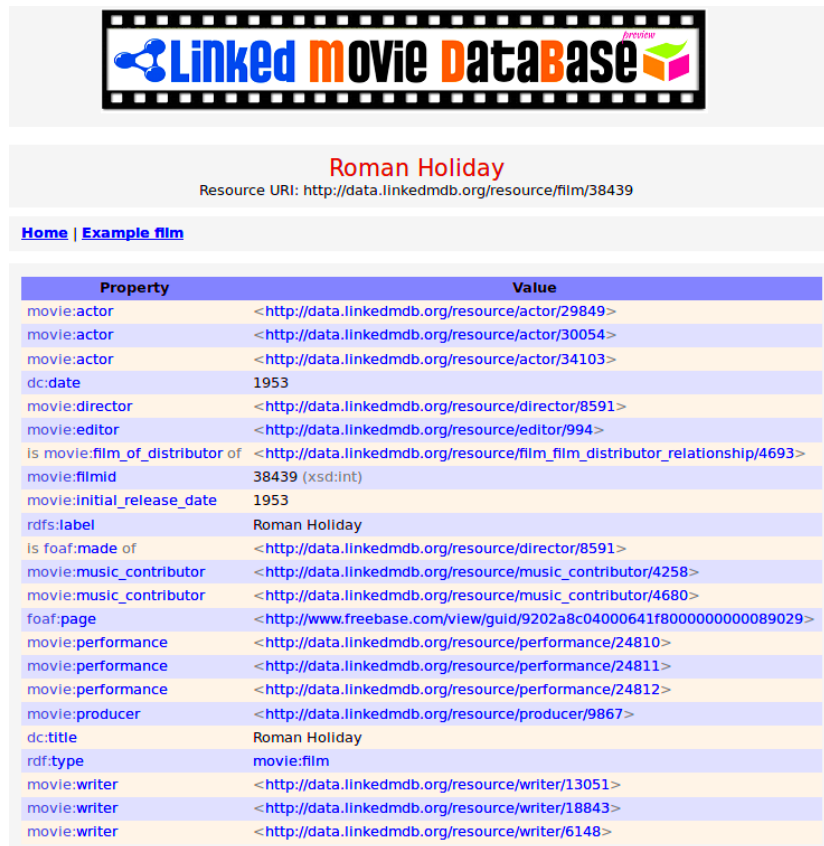
⁷<http://www.w3.org/DesignIssues/LinkedData.html>, retrieved 2013-10-05.

⁸<http://www.w3.org/standards/semanticweb/data>, retrieved 2013-10-07.

⁹<http://microformats.org/>, retrieved 2013-10-05.

¹⁰<http://webdatacommons.org/>, retrieved 2013-10-05.

¹¹<http://commoncrawl.org/>, retrieved 2013-10-06.



The screenshot shows the 'Roman Holiday' page on the Linked Movie Database. At the top is the logo for 'Linked movie DataBase'. Below it, the title 'Roman Holiday' is displayed in red, with the Resource URI: <http://data.linkedmdb.org/resource/film/38439>. There are links for 'Home' and 'Example film'. The main content is a table with two columns: 'Property' and 'Value'. The table lists various properties such as 'movie:actor', 'dc:date', 'movie:director', 'movie:editor', 'is movie:film_of_distributor_of', 'movie:filmid', 'movie:initial_release_date', 'rdfs:label', 'is foaf:made_of', 'movie:music_contributor', 'foaf:page', 'movie:performance', 'movie:producer', 'dc:title', 'rdf:type', and 'movie:writer', each with its corresponding value.

Property	Value
movie:actor	< http://data.linkedmdb.org/resource/actor/29849 >
movie:actor	< http://data.linkedmdb.org/resource/actor/30054 >
movie:actor	< http://data.linkedmdb.org/resource/actor/34103 >
dc:date	1953
movie:director	< http://data.linkedmdb.org/resource/director/8591 >
movie:editor	< http://data.linkedmdb.org/resource/editor/994 >
is movie:film_of_distributor_of	< http://data.linkedmdb.org/resource/film_film_distributor_relationship/4693 >
movie:filmid	38439 (xsd:int)
movie:initial_release_date	1953
rdfs:label	Roman Holiday
is foaf:made_of	< http://data.linkedmdb.org/resource/director/8591 >
movie:music_contributor	< http://data.linkedmdb.org/resource/music_contributor/4258 >
movie:music_contributor	< http://data.linkedmdb.org/resource/music_contributor/4680 >
foaf:page	< http://www.freebase.com/view/guid/9202a8c04000641f800000000089029 >
movie:performance	< http://data.linkedmdb.org/resource/performance/24810 >
movie:performance	< http://data.linkedmdb.org/resource/performance/24811 >
movie:performance	< http://data.linkedmdb.org/resource/performance/24812 >
movie:producer	< http://data.linkedmdb.org/resource/producer/9867 >
dc:title	Roman Holiday
rdf:type	movie:film
movie:writer	< http://data.linkedmdb.org/resource/writer/13051 >
movie:writer	< http://data.linkedmdb.org/resource/writer/18843 >
movie:writer	< http://data.linkedmdb.org/resource/writer/6148 >

Figure 3: Linked MDB page about the movie “Roman Holiday”.¹² Structured data is encoded as RDF and published adhering to Linked Data standards.

- ② The terminology Linked Data describes a popular set of principles for publishing RDF data on the Web. Currently more than 62 billion triples in 920 datasets are published as Linked Data.¹³ Following Linked Data principles, HTTP links connect entities (instead of Web pages), thereby associating “data elements” directly. Furthermore, links between entities are typed, which enables a characterization of the relationship. Prominent examples of Linked Data datasets include DBpedia¹⁴ and LinkedMDB.¹⁵ The former comprises structured data extracted from Wikipedia,¹⁶ while the latter contains data about movies.

Example 2

Consider Figure 3: a set of triples describes the movie “Roman Holiday”. These triples actually capture the data semantics, i.e., the underlying meaning of the data – in contrast to the corresponding HTML page in Figure 2. More precisely, each triple assigns the sub-

¹²<http://data.linkedmdb.org/page/film/38439>, retrieved 2013-10-05.

¹³<http://stats.lod2.eu/>, retrieved 2013-10-05.

¹⁴<http://dbpedia.org/>, retrieved 2013-10-05.

¹⁵<http://linkedmdb.org/>, retrieved 2013-10-05.

¹⁶<http://www.wikipedia.org/>, retrieved 2013-10-05.

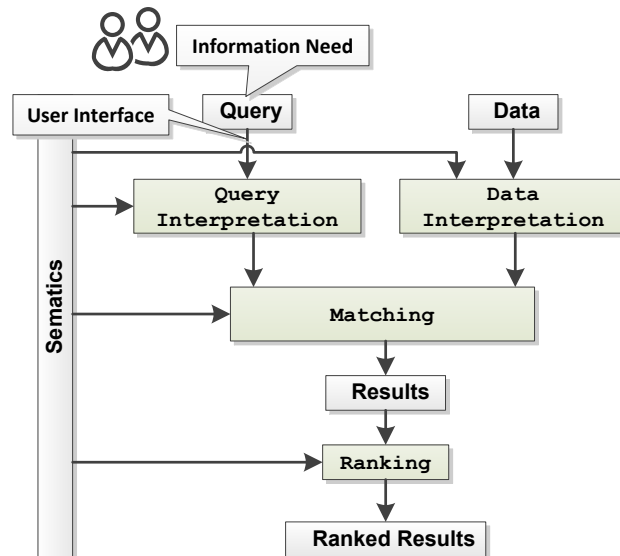


Figure 4: The semantic search process [160, 7].

ject (entity “Roman Holiday”) an object via a predicate. For example, the triple $\langle \text{film:38439}, \text{rdf:type}, \text{movie:film} \rangle$ states that the entity “Roman Holiday” (identified with `film:38439`) has `movie:film` via `rdf:type` assigned.^a

^aThe URI prefix `film` and `movie` stands for `http://data.linkedmdb.org/resource/film` and `http://data.linkedmdb.org/resource/movie`, respectively. Further, the URI prefix `rdf` stands for `http://www.w3.org/1999/02/22-rdf-syntax-ns#`.

- ③ Last, schemata on the Web provide reusable vocabularies for Web data. A well-known example is Schema.org¹⁷ – an effort by major search engine providers (Google, Bing, Yahoo, and Yandex) to create and maintain schema information for a structured Web page markup. A Schema.org version adhering to Linked Data principles is also available.¹⁸

1.1.2 Semantic Search

1.1.2.1 Overview

Search is commonly known as an end-user paradigm that aims at satisfying information needs via simple user interfaces/access mechanisms [160]. That is, a user expresses an information need as a query via an interface and the engine attempts to discover data elements, which are assumed to satisfy that need.

However, because of query/data ambiguities, discovering such relevant elements may be hard. Semantic data as well as semantic query representations target this issue and thereby help to improve Web search significantly. *Semantic search* can be defined as follows:

¹⁷<http://schema.org/>, retrieved 2013-10-05.

¹⁸<http://schema.rdfs.org/>, retrieved 2013-10-05.

» Definition 2: Semantic Search (Informal) [160]

“Semantic search [...] makes use of explicit semantics to solve core search tasks, i.e., to use semantics for interpreting query and data, matching query against data and ranking results.”, see [160].

A generic semantic search process is illustrated in Figure 4. Intuitively, a semantic search process first determines possible query/data interpretations. Various interpretations are caused by ambiguities in query or data. Next, depending on the chosen query/data interpretation, query results are evaluated by matching query constraints to data elements. Last, every result is associated with a ranking score, which captures the relevance of that result with regard to the query and user intent. This way, an interface may present top-ranked results first and allow users to quickly satisfy their information needs [160, 7].

Semantics – as captured by Web data – are exploited at every step of the search process, in order to determine the user’s information need and find data elements that match this need best [160, 7].

1.1.2.2 Problems

Semantic search engines oftentimes produce *vary large result sets* due to multiple query/data interpretations as well as extensive datasets on the Web. Large result sets, in turn, make result ranking essential for discovering relevant results and satisfying an information need. At the same time, system efficiency/responsiveness is crucial for many application domains. Most notably, for end-user systems responsiveness is a key requirement. Thus, semantic search systems must address the problem on *how to compute large ranked result sets and report the most important (top-ranked) results as soon as possible*:

⚡ Problem 1: Rank-aware Query Processing

Compute ranked results efficiently and report top-ranked results as soon as possible.

Another way to cope with large result sets is to relax the requirement for exact and complete results. In fact, many semantic search applications are end-user oriented. Here, users frequently omit results and only view a small fraction of the entire result set. Thus, semantic search systems should allow for a *trade-off between result accuracy/completeness and computation time*:

⚡ Problem 2: Approximate Query Processing

Compute large result sets efficiently by allowing to trade off result accuracy and result completeness for computation time.

Semantic search has received much attention and poses many other problems. However, in this thesis we are solely concerned with Problem 1 and Problem 2. The interested reader may see the comprehensive surveys [160, 7]. Note, we also discuss the scope of this thesis in Section 1.3.2.

IMDb Find Movies, TV shows, Celebrities and more... All Q

Movies TV News Showtimes Community IMDbPro Apps

Highest Rated Feature Films With Title Matching "Holiday"

Sort by: [MOVIEmeter](#) | [A-Z](#) | [User Rating](#) ▼ | [Num Votes](#) | [US Box Office](#) | [Runtime](#) | [Year](#) | [US Release Date](#)

- Holiday Road (2012)** [Add to Watchlist](#)
 ★★★★★★☆☆ 8.2/10
 A comedy anthology film in which thirteen filmmakers each take on a different American holiday.
 Dir: Aaron Arendt, Todd Berger With: Meggan Amos, Jessica Antonucci, Lesley Bargar Suter
 Comedy
- Roman Holiday (1953)** [Add to Watchlist](#)
 ★★★★★★☆☆ 8.1/10
 A bored and sheltered princess escapes her guardians and falls in love with an American newsman in Rome.
 Dir: William Wyler With: Gregory Peck, Audrey Hepburn, Eddie Albert
 Comedy | Drama | Romance 118 mins. [Approved](#)
- Holiday (1938)** [Add to Watchlist](#)
 ★★★★★★☆☆ 7.7/10
 A young man falls in love with a girl from a rich family. His unorthodox plan to go on holiday for the early years of his life is met with skepticism by everyone except for his fiancée's eccentric sister and long suffering brother.
 Dir: George Cukor With: Katharine Hepburn, Cary Grant, Doris Nolan
 Comedy | Romance 95 mins. [Approved](#)
- Johnny Holiday (1949)** [Add to Watchlist](#)
 ★★★★★★☆☆ 7.7/10
 Young street tough sent to a reform farm is torn between friends from his past and those who are trying to help him change his life.
 Dir: Willis Goldbeck With: William Bendix, Stanley Clements, Hoagy Carmichael
 Crime | Drama 92 mins.
- On Holiday (2010)** [Add to Watchlist](#)
 ★★★★★★☆☆ 7.6/10
 Dir: Brian McGuire With: Whitmer Thomas, Jennifer June Ross, Tipper Newton
 Comedy 85 mins.

Figure 5: Search for high ranking movies with keyword “holiday” on IMDB Web page. Ranking scores are based on user ratings.¹⁹

In the next paragraphs, let us briefly introduce the fields of research concerned with above problems.

1.1.1.2.3 Problem 1: Rank-aware Query Processing

Rank-aware query processing, so-called top-k query processing, is a very active research area in the database (DB) community [95]. Here, the goal is to *compute top-ranked query results, without materializing the entire result set*. Consider the example:

Example 3

Figure 5 depicts a ranked list of movies for the keyword query “holiday”. Computing this list could be done in multiple ways. The naïve solution would be:

- ① Materialize a list of entities, which have the keyword “holiday” associated.
- ② Further, materialize a second list of movie entities and join both lists.
- ③ Sort the movie entities in the joined list based on their user ratings.

Computing ranked results in such a manner is highly inefficient [95]. Above solution (Example 3) requires three lists of intermediate results: an entity list for keyword query “holiday”, another list of movie entities, and the joined entity list. In fact, the entire joined list must be materialized before it can be sorted and reported. This is because sorting is a blocking operation, i.e., the results can only be sorted if all results have been computed. Given queries with large result sets, this procedure leads to prohibitive join and sort operations [95].

Rank-aware query processing strategies target this problem by embedding ranking within the query processing [95]. More specifically, these strategies allow search engines to *iteratively compute ranked results*, without the need to fully materialize the result set. For this, ranking scores of already computed results are used to estimate whether or not “higher-ranked” results could be found by computing further results [95]. This can lead to drastic runtime efficiency gains – as we will show in Chapter 3 and Chapter 5.

1.1.2.4 Problem 2: Approximate Query Processing

We employ approximate query processing (AQP) to target the above Problem 2. AQP comprises a set of techniques, which allow to save time by computing approximate instead of exact results [112]. Such approximation techniques frequently utilize compact data synopses, in order to produce query results that are “similar” to exact results. Such synopses may be based on random samples, wavelets, or histograms [112]. Traditionally, AQP was employed in the database community for aggregate queries featuring functions like `sum()` or `avg()` [112].

However, we use AQP in context with rank-aware query processing. Consider the following example:

Example 4

A user is searching for top-ranking movies with keyword “holiday” in Example 3. However, the user probably does not care about the precise ranking values. That is, she probably would not mind if the *ranking values were approximated* – as long as the result order remains similar.

Further, the user will first look at the top-ranked movies and only later (if necessary) consider additional results. Thus, a system *could incrementally compute query results* and approximate query results with low ranking.

Generally speaking, we consider AQP in terms of two dimensions:

① Incremental Result Computation

First, we consider incremental result computation as one AQP dimension. That is, query results are processed step-by-step via a pipeline of operations. This way, initial/approximated results can be obtained very early and can be refined as needed, see Section 5.2.

② Rank-aware Approximation

Second, rank-aware approximation techniques resemble another AQP dimension. More specifically, we target the problem of approximating the

¹⁹Web search on <http://www.imdb.com/search/>. Web page retrieved at 2013-10-05.

ranking value for a particular result. Intermediate results, which are not likely to contribute to the final top-ranked results are discarded during query processing, see Section 5.3.

1.2 WEB DATA CHARACTERISTICS

Web data *features several characteristics* that split Problem 1 and Problem 2 into several research questions (presented in the next section). In the following, let us discuss key characteristics, which we faced in our work.

1.2.1 Characteristic 1: Schemaless Data

Web data comes with great flexibility in terms of its associated schema. In fact, data on the Web often has little or no schema information. In particular, entities frequently have no RDFS [29] or Web Ontology Language (OWL) [82, 8] ontology²⁰ assigned. Thus, entities could only be described via instance data and not in a formal manner, e.g., by means of class or property definitions.

1.2.2 Characteristic 2: Hybrid Data

Web data contains different types of entity descriptions: text-rich, structured, and formal descriptions [160]. The former refers to unstructured/textual descriptions associated with entities. Structured descriptions, however, comprise attributes and relations as well as classes. Generally speaking, these descriptions feature data elements, which adhere to a pre-defined schema. Last, formal entity descriptions may be seen as a special kind of structured descriptions, which use highly expressive representations such as OWL [82, 8] or F-Logic [101].

Example 5

A movie entity could have a (lengthy) textual description via a comment or a plot attribute, see Example 3. Furthermore, that entity may have relations (e.g., starring) or classes (e.g., movie or actor). Last, the entity could have a formal description stating that every movie must have at least one actor assigned via the starring relation.

From a general point of view, entities on the Web contain *unstructured* and *structured* descriptions – commonly known as *hybrid data*. In the DB and Information Retrieval (IR) community challenges associated with hybrid data are known as DB&IR integration [165]. This kind of Web data will be queried with constraints that match structured as well as unstructured data elements [171]. Consequently, we will refer to such queries as *hybrid queries*.

²⁰<http://www.w3.org/standards/semanticweb/ontology>, retrieved 2013-10-08.

1.2.3 Characteristic 3: Distributed and Low-volume Data

Web data is usually highly distributed over a large space of low-volume data sources. That is, data is not centrally stored and managed in few large databases, but located in various small sources with restricted capabilities [97, 105]. This development is fostered by the Linked Data principles as well as semantic annotations standards.

① *Low-volume Data with Simple Access*

Data sources frequently comprise very few entities and small entity descriptions. Following Linked Data principles, every HTTP URI identifies a “virtual data source” [105]. Access to this source is done via HTTP operations. For instance, HTTP GET is used for retrieving the entity’s description and HTTP DELETE for deletion of the entity (identified with that URI) [153]. For a complete listing of supported operations see [153].

② *Distributed Data*

The Web of data does not restrict or control data publishing: everybody may publish and interlink data [97]. Such a lack of a “controlled authorship” results in a high distribution and wide range of Web data sources. For instance, there are currently more than 920 Linked Data datasets publicly available.²¹ Note, the number of sources increases drastically, if one regards every entity as its own data source [105].

Web data features many other characteristics, which may lead to further research questions. However, in this thesis we focus on the above characteristics. See also Section 1.3.2 for the scope of this thesis.

1.3 RESEARCH QUESTIONS AND SCOPE

1.3.1 Research Questions

Based on Problem 1 and Problem 2, our overall research question is:

Overall Research Question

How to allow for rank-aware and approximate query processing over Web data?

Given Web data characteristics in Section 1.2, the overall question breaks down into several research questions, which we target in Chapter 3, Chapter 4, and Chapter 5. An overview of our problems, the addressed research questions as well as Web data characteristics is depicted in Figure 6.

²¹<http://stats.lod2.eu/>, retrieved 2014-02-01.

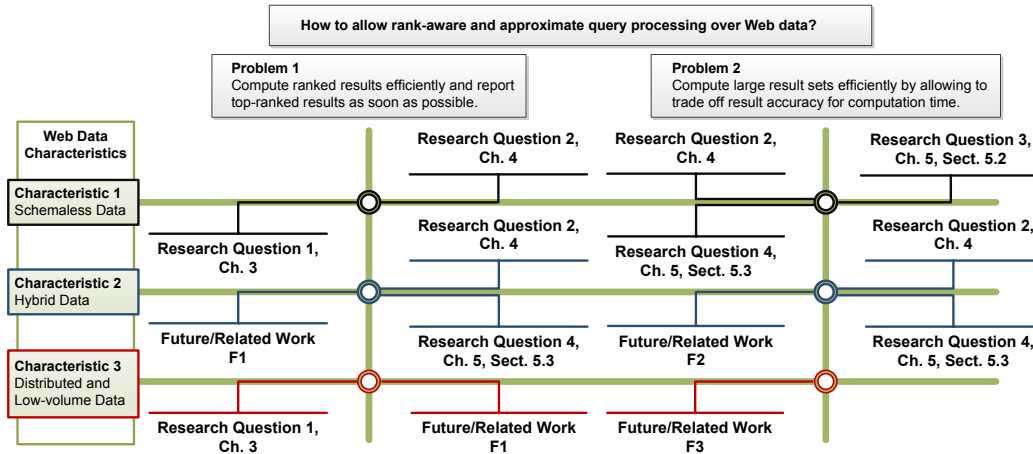


Figure 6: Overview of problems, Web data characteristics, and research questions, which are address in this thesis.

🔗 Research Question 1

How to enable top-k query processing on highly distributed, schemaless Web data?

Research Question 1 is driven by the *distributed nature* of Web data (Characteristic 3) and aims at allowing rank-aware query processing in such a context (Problem 1). More precisely, our task is to process joins over data from distributed sources in such a manner that high-ranked results are reported first. We investigate top-k join processing techniques for this task and extend previous works [55, 116, 144, 170] in Chapter 3 to match our *distributed and low-volume Web data* (Characteristic 3).

Moreover, our approach does not require schema information. Thus, we allow for *schemaless* Web data (Characteristic 1) to be processed.

🔗 Research Question 2

How to allow for efficient and effective selectivity estimates on hybrid, schemaless Web data?

Driven by *schemaless* (Characteristic 1) and *hybrid* (Characteristic 2) Web data, we address Research Question 2 as a key problem in Chapter 4. In this chapter, we target the question: How to efficiently and effectively predict the result size of queries over Web data? More specifically, our selectivity estimation approach calculates the cardinality of queries, which may comprise structured as well as unstructured query constraints. While the former query constraint matches structured entity descriptions, the latter constraint matches keywords in unstructured entity descriptions.

Selectivity estimation plays a crucial role for query optimization, as it allows to integrate join operators in physical query plans [53]. Further, we exploit selectivity estimation to support our rank-aware approximate query processing in Section 5.3 (see Research Question 4).

Last, we present two complementary forms of approximate query processing: (1) incremental result approximation in Research Question 3 and (2) rank-aware approximation in Research Question 4.

↳ Research Question 3

How to enable approximate and incremental query processing on schemaless Web data?

Incremental query processing directly reflects Problem 2, because initial (approximated) results can be reported very early. Intuitively, query results are computed step-by-step and results can be returned at any time during this procedure. We present a pipeline-based approach targeting this question in Chapter 5, Section 5.2. In particular, our approach *does not require any schema information* throughout the pipeline (Characteristic 1).

↳ Research Question 4

How to enable approximate top-k query processing for hybrid queries over schemaless Web data?

Research Question 4 aims at approximating query results with regard to their associated ranking position. That is, instead of approximating all query bindings, we specifically approximate the low-ranked results. The former is addressed by the aforementioned Research Question 3. We present a novel approach for such a rank-aware approximation in Chapter 5 (see Section 5.3), which works with *schemaless* (Characteristic 1) as well as *hybrid queries and hybrid Web data* (Characteristic 2).

1.3.2 Scope of this Thesis

Semantic search has been addressed by many other works before [160, 7]. The interested reader may see [158] for a complete semantic search process. In particular, [104] aims at an efficient query processing over hybrid Web data. In fact, [104] presents query processing as well as indexing techniques, which are complementary to the approaches in this thesis. Moreover, [79] targets the effectiveness of ranking techniques over hybrid Web data. In contrast, we will not focus on effectiveness issues of different ranking techniques, but simply consider a ranking function to be given.

Generally speaking, we solely target the above research questions in Chapter 3, Chapter 4, and Chapter 5. With regard to those questions, this thesis provides several novel contributions – as we will outline in the next section.

1.4 CONTRIBUTIONS

With regard to the aforementioned research questions, this thesis provides the following contributions:

☞ Contribution for Research Question 1

Top-k join processing over Linked Data.

Existing work on top-k processing in the DB community targets scenarios, where data is centrally stored and managed, or where data is located at few/large Web databases [51, 95]. In contrast, Web data is highly distributed over a large space of small data sources – most notably, Web data sources adhering to the Linked Data principles.

In Chapter 3, we will show how to extend well-known top-k processing techniques [51, 95] to the Web of Linked Data sources. Chapter 3 is based on our previous publication [2] and targets the *schemaless* (Characteristic 1) as well as *highly distributed* nature (Characteristic 3) of Web data.

☞ Contribution for Research Question 2

Selectivity estimation for hybrid and schemaless Web data.

Based on our publication [3], we present a novel selectivity estimation approach for hybrid schemaless Web data in Chapter 4. For this, we combine a template-based Bayesian network with string synopses. More specifically, we extend existing work for selectivity estimation over relational DBs [60, 162] to effectively estimate queries over *schemaless* Web data (Characteristic 1). Furthermore, we estimate the selectivity of keyword queries over textual entity descriptions by means of string synopses (Characteristic 2).

☞ Contribution for Research Question 3

Approximate and incremental query processing over Web data.

Based on our work in [1], we give an incremental query processing approach in Chapter 5, see Section 5.2. For this, we decompose the query processing into four sequential phases that operate on different data synopses, which are well-suited for *schemaless* Web data (Characteristic 1). Each phase produces an intermediate/approximate result, which is refined (if necessary) by the subsequent phase. This way, a system may report initial/approximate early, if the information need does not require correct and complete results, respectively.

Moreover, we propose two novel approximate join processing techniques that are employed in this processing pipeline. On the one hand, we propose approximate structure matching that operates on bloom filters. On the other hand, we propose structure-based result refinement, which exploits a compact data synopsis (the so-called structure index) for approximate join processing.

☞ Contribution for Research Question 4

Approximate top-k query processing for hybrid queries over Web data.

For Research Question 4, we investigate rank-aware approximate query processing in Chapter 5, see Section 5.3. This chapter is based on a work, which

is published in [4]. Our approach estimates how likely an intermediate result leads to a final top-k result – intermediate results below a given threshold will be pruned. For this, we employ work on selectivity estimation (as presented in Chapter 4) as well as techniques from the field of Bayesian statistics. More precisely, our statistics are learned in a pay-as-you-go manner during query processing. In particular, these statistics allow for keyword queries, thereby enabling an effective search over *hybrid* Web data (Characteristic 2). Moreover, our statistics are very lightweight, which enables a system to efficiently maintain its indexes. Efficient maintenance is a crucial advantage with regard to the frequently changing Web data.

1.5 OUTLINE

The remainder of this thesis comprises six chapters, which discuss Contributions 1 - 4 and aim at Research Questions 1 - 4.

② *Chapter 2 – Foundations*

Chapter 2 provides preliminaries for our approaches in Chapter 3, Chapter 4, and Chapter 5. In particular, we introduce our data/query model. Furthermore, we outline basic query processing strategies as well as introduce top-k processing techniques from the DB community.

③ *Chapter 3 – Rank-aware Query Processing*

In Chapter 3, we present a novel approach for top-k join processing over Linked Data. For this, we extend traditional top-k techniques to the Web of Linked Data sources. That is, we specifically target distributed and low-volume data sources.

④ *Chapter 4 – Selectivity Estimation*

We introduce a selectivity estimation approach for queries over schemaless, hybrid Web data in Chapter 4. That is, our work allows query constraints that match structured as well as unstructured entity descriptions.

⑤ *Chapter 5 – Approximate Query Processing*

Chapter 5 features two approaches for approximate query processing over Web data. First, we discuss a pipeline of operations for an incremental processing of queries in Section 5.2. Second, we propose a rank-aware query result approximation in Section 5.3. These approaches are complementary to each other, i.e., a system may apply both approaches as means to compute approximated ranked results.

⑥ *Chapter 6 – Conclusion*

Last, we summarize our contributions and results in Chapter 6. Further, we give an outlook on important future work.

FOUNDATIONS

FOUNDATIONS

In this chapter, we discuss the preliminaries for the remainder of the thesis. In particular, we present our data/query model. The former specifies the data representation and the latter defines a query language over this data, see Section 2.1. In Section 2.2, we outline our result model and basic query processing techniques. Last, we introduce top-k query processing strategies in Section 2.3 – a particular kind of query processing that is “rank-aware”.

2.1 DATA AND QUERY MODEL

In recent years, RDF became a standard for describing entities on the Web. Thus, as a particular form of Web data (see Definition 1, p. 4), we use RDF [147] as data model. RDF data may be conceived as a data graph that connects and describes entities. More precisely, RDF data constitutes a set of *triples* $\{\langle s, p, o \rangle\}$ forming a data graph, see Figure 7. Every triple describes a particular entity (the *subject*) s through a *predicate/object* pair: p/o .

The standard language for querying RDF is SPARQL [17]. In this thesis, we restrict our attention to a key fragment of SPARQL: basic graph pattern (BGP) queries. See Section 2.1.2 for further details on the employed query model.

2.1.1 Structured and Unstructured Data

»» Definition 3: RDF Graph, RDF Triple

Let ℓ_a and ℓ_r denote a set of attribute and relation labels. RDF data may be seen as a directed labeled graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \ell_a, \ell_r)$, where \mathcal{V} is the disjoint union $\mathcal{V} = \mathcal{V}_E \uplus \mathcal{V}_A \uplus \mathcal{V}_C$, with \mathcal{V}_E as entity nodes, \mathcal{V}_A as attribute value nodes, and \mathcal{V}_C as class nodes.

Edges (*triples*) $\mathcal{E} = \mathcal{E}_R \uplus \mathcal{E}_A$ are a disjoint union of relation edges \mathcal{E}_R and attribute edges \mathcal{E}_A . Relation edges connect entity nodes: $\langle s, r, o \rangle \in \mathcal{E}_R$ iff $s \in \mathcal{V}_E$, $r \in \ell_r \uplus \text{type}$, and $o \in \mathcal{V}_E \uplus \mathcal{V}_C$. Attribute edges connect an entity with an attribute value: $\langle s, a, o \rangle \in \mathcal{E}_A$ iff $s \in \mathcal{V}_E$, $o \in \mathcal{V}_A$, and $a \in \ell_a$.

The “special” relation edge $\langle s, \text{type}, c \rangle \in \mathcal{E}_R$, $s \in \mathcal{V}_E$, and $c \in \mathcal{V}_C$, models that entity s belongs to class c .

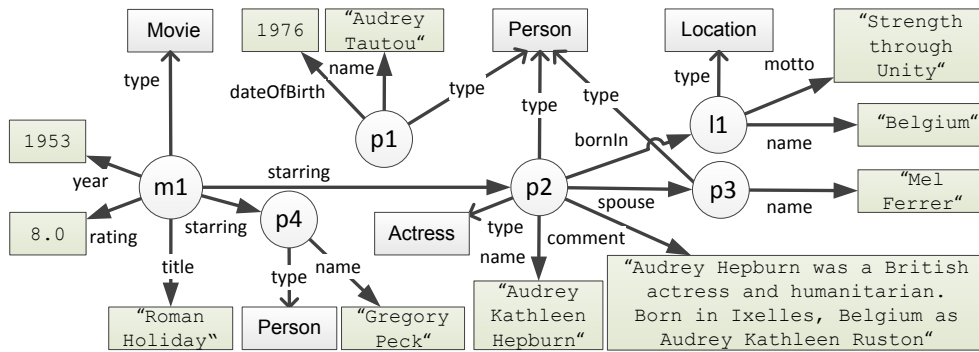


Figure 7: RDF graph that captures information about Audrey Hepburn and her movie “Roman Holiday”.

Example 6

Example data is depicted in Figure 7. More specifically, the example data graph has an entity set $\mathcal{V}_E = \{m_1, p_1, p_2, p_3, p_4, l_1\}$ and a set of attribute value nodes $\mathcal{V}_A = \{\text{“Roman Holiday”, “Audrey Kathleen Hepburn”, ...}\}$. Attribute edges are given by $\mathcal{E}_A = \{\text{name, title, ...}\}$ and relation edges are $\mathcal{E}_R = \{\text{starring, spouse, ...}\}$. Further, entities are assigned to classes: $\mathcal{V}_C = \{\text{Movie, Person, Actress, Location}\}$.

Unstructured Data. Many RDF graphs are *text-rich*, i.e., they contain large amounts of textual data. More formally, attribute value nodes in \mathcal{V}_A oftentimes comprise large text values. For instance, our data graph in Figure 7 features texts for attributes *comment*, *motto*, or *title*.

Generally speaking, structured RDF data frequently has text via predicates such as `rdfs:comment` or `dc:description`.²² Well-known examples are the DBpedia²³ or IMDB²⁴ dataset. Furthermore, unstructured Web documents are frequently annotated with structured data using, e.g., RDFa or Microformats.²⁵ Such interlinked documents can be seen as an RDF graph with documents as objects. For instance, the Wikidata project²⁶ recently introduced structured data to the Wikipedia corpus.

Structured Data. At the same time, RDF data also contains structured data in form of attribute (\mathcal{E}_A) and relation edges (\mathcal{E}_R), as well as classes in \mathcal{V}_C . With regard to our example in Figure 7, the graph contains, e.g., the *Movie* class, the *starring* relation, or the *title* attribute as structured data. Notice, for simplicity we omit additional RDFS [29] features such as predicate or class hierarchies. Fur-

²²Prefix `rdfs` and `dc` stand for <http://www.w3.org/2000/01/rdf-schema#> and <http://purl.org/dc/elements/1.1/>, respectively.

²³<http://dbpedia.org>, retrieved 2013-10-30.

²⁴<http://www.linkedmdb.org>, retrieved 2013-10-30.

²⁵<http://www.webdatacommons.org>, retrieved 2013-10-30.

²⁶<http://www.wikidata.org>, retrieved 2013-10-30.

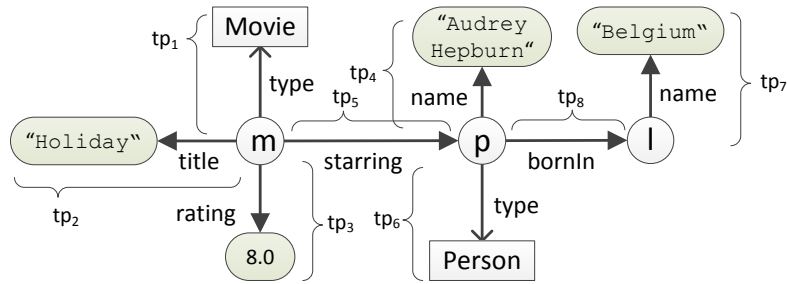


Figure 8: Hybrid query graph Q , which is asking for movies with title “Holiday”, starring “Audrey Hepburn” etc. Data is given in Figure 7. Query Q comprises eight triple patterns: $Q = \{tp_1, \dots, tp_8\}$.

```

1 PREFIX ex: <http://example.org/>
2 PREFIX xs: <http://www.w3.org/2001/XMLSchema#>
3 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
4
5 SELECT *
6 WHERE
7 {
8   ?m ex:title ?t .
9   FILTER contains(?t , "Holiday") .
10  ?m ex:rating "8.0"^^xs:double .
11  ?m rdf:type ex:Movie .
12  ?m ex:starring ?p .
13  ?p ex:name ?n1 .
14  FILTER contains(?n1 , "Audrey Hepburn") .
15  ?p rdf:type ex:Person .
16  ?p ex:bornIn ?l .
17  ?l ex:name ?n2 .
18  FILTER contains(?n2 , "Belgium") .
19 }

```

Listing 1: Hybrid query from Figure 8 written in SPARQL 1.1 syntax.

then note that more expressive data representations, e.g., OWL [82, 8], introduce even more schema data.

Hybrid Data. For this thesis, we focus on the *mixture of structured and unstructured data* in RDF graphs – we refer to this kind of data as *hybrid data*. Moreover, if an attribute value $o \in \mathcal{V}_A$ contains text, we conceive it as a *bag-of-words*. We say that a vocabulary \mathcal{W} comprises all such bags-of-words in \mathcal{V}_A .

Example 7

For instance, in Figure 7 we would regard attribute value “Roman Holiday” as a bag-of-words: {“Roman”, “Holiday”}. Further, we have a vocabulary $\mathcal{W} = \{“Roman”, “Holiday”, “Audrey”, “Kathleen”, \dots\}$, which captures all such words in Figure 7.

2.1.2 Structured and Unstructured Queries

Next, let us introduce a query model well-suited for RDF graphs following Definition 3. We employ a core part of SPARQL [17]: basic graph patterns (BGPs) queries. Furthermore, we present a corresponding result model together with its formal semantics.

BGP Queries and Result Model. BGP queries comprise a conjunction of *triple patterns*. We use a particular type of BGP queries: *hybrid queries*. Hybrid queries can comprise some patterns that match structured data, while other patterns can match unstructured texts.

» Definition 4: Hybrid Query, Triple Pattern

Given a data graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \ell_a, \ell_r)$, a *hybrid query* \mathcal{Q} is a directed labeled graph $\mathcal{G}^{\mathcal{Q}} = (\mathcal{V}^{\mathcal{Q}}, \mathcal{E}^{\mathcal{Q}})$, where $\mathcal{V}^{\mathcal{Q}}$ is the disjoint union $\mathcal{V}^{\mathcal{Q}} = \mathcal{V}_V^{\mathcal{Q}} \uplus \mathcal{V}_C^{\mathcal{Q}} \uplus \mathcal{V}_K^{\mathcal{Q}}$, with $\mathcal{V}_V^{\mathcal{Q}}$ as a set of variable nodes, $\mathcal{V}_C^{\mathcal{Q}} = \mathcal{V} = \mathcal{V}_E \uplus \mathcal{V}_A \uplus \mathcal{V}_C$ as a set of constants, and $\mathcal{V}_K^{\mathcal{Q}}$ as a set of user-defined keywords. Edges $\mathcal{E}^{\mathcal{Q}}$ are called *triple patterns*, with each pattern adhering to:

$$\text{tp} = \langle s, p, o \rangle$$

where $s \in \mathcal{V}_V^{\mathcal{Q}} \uplus \mathcal{V}_E$, $p \in \ell_a \uplus \ell_r \uplus \mathcal{V}_V^{\mathcal{Q}}$, and $o \in \mathcal{V}_V^{\mathcal{Q}} \uplus \mathcal{V}_A \uplus \mathcal{V}_C \uplus \mathcal{V}_K^{\mathcal{Q}}$.

As a shorthand, we will sometimes write a query \mathcal{Q} as a *set of its triple patterns*: $\mathcal{Q} = \{\text{tp}_i\}$. For example, query $\mathcal{Q} = \{\text{tp}_1, \dots, \text{tp}_8\}$ in Figure 8.

For simplicity, we define a keyword node as a single word occurring in the text. That is, a keyword is one element from a bag-of-words representation of an attribute node.

» Example 8

An example query \mathcal{Q} is shown in Figure 8. Query \mathcal{Q} features eight triple patterns, which match structured and unstructured data, respectively. For instance, the triple pattern

$$\text{tp}_1 = \langle m, \text{starring}, p \rangle$$

has two variable nodes (m and p) and matches the `starring` relation in Figure 7. Furthermore, the pattern

$$\text{tp}_2 = \langle m, \text{title}, \text{"Holiday"} \rangle$$

has a keyword “Holiday”, and matches the attribute node “Roman Holiday”. Query \mathcal{Q} can also be written in SPARQL syntax – as shown in Listing 1.

Corresponding to edge types, ℓ_a , ℓ_r , and `type` in Definition 3, we distinguish four kinds of query patterns:

» **Definition 5: Class, Relation, Attribute, and String Triple Pattern**

A query \mathcal{Q} can comprise:

- *Class Pattern*
 $tp = \langle s, \text{type}, o \rangle, s \in \mathcal{V}_V^{\mathcal{Q}} \uplus \mathcal{V}_E$ and $o \in \mathcal{V}_C$.
- *Relation Pattern*
 $tp = \langle s, r, o \rangle, s \in \mathcal{V}_V^{\mathcal{Q}} \uplus \mathcal{V}_E, o \in \mathcal{V}_C^{\mathcal{Q}} \uplus \mathcal{V}_V^{\mathcal{Q}}$, and $r \in \ell_r$.
- *Attribute Pattern*
 $tp = \langle s, a, o \rangle, s \in \mathcal{V}_V^{\mathcal{Q}} \uplus \mathcal{V}_E, o \in \mathcal{V}_C^{\mathcal{Q}} \uplus \mathcal{V}_V^{\mathcal{Q}}$, and $a \in \ell_a$.
- *String Pattern*
 $tp = \langle s, a, w \rangle, s \in \mathcal{V}_V^{\mathcal{Q}} \uplus \mathcal{V}_E, w \in \mathcal{V}_K^{\mathcal{Q}}$, and $a \in \ell_a$.

Notice, we will sometimes refer to class, relation, and attribute pattern as *structured query pattern*.

Result Model. A result (*binding*) for a query \mathcal{Q} is defined as:

» **Definition 6: Binding**

A *binding* b for a query \mathcal{Q} is a vector (t_1, \dots, t_n) of triples, such that each triple t_i *matches* (defined in Definition 8 and Definition 9) exactly one pattern tp_i in \mathcal{Q} and triples in b form a subgraph of \mathcal{G} .

Via the matching of patterns in \mathcal{Q} to triples, b *binds* variables to nodes in the data. Formally, for binding b there is a function $\mu_b : \mathcal{V}_V^{\mathcal{Q}} \mapsto \mathcal{V}$, mapping every variable in \mathcal{Q} to a node in \mathcal{V} .

Notice, we will use the terms *result* and *binding* synonymously in the following. Furthermore, the *result set* for a query \mathcal{Q} is denoted as \mathbf{B} .

 **Example 9**

Continuing Example 8, one possible binding (result) for \mathcal{Q} is:

$b = (t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8)$, where

- triple $t_1 = \langle m_1, \text{type}, \text{Movie} \rangle$ matches tp_1 ,
- triple $t_2 = \langle m_1, \text{title}, \text{"Roman Holiday"} \rangle$ matches tp_2 ,
- triple $t_3 = \langle m_1, \text{rating}, 8.0 \rangle$ matches tp_3 ,
- triple $t_4 = \langle p_2, \text{name}, \text{"Audrey Kathleen Hepburn"} \rangle$ matches tp_4 ,
- triple $t_5 = \langle m_1, \text{starring}, p_2 \rangle$ matches tp_5 ,
- triple $t_6 = \langle p_2, \text{type}, \text{Person} \rangle$ matches tp_6 ,
- triple $t_7 = \langle l_1, \text{name}, \text{"Belgium"} \rangle$ matches tp_7 , and
- triple $t_8 = \langle p_2, \text{bornIn}, l_1 \rangle$ matches tp_8 .

Function μ_b binds all variables in query \mathcal{Q} . That is, $\mu_b(m) = m_1$, $\mu_b(p) = p_2$, and $\mu_b(l) = l_1$.

During query processing *partial* bindings, which feature some patterns with no matching triples, will occur:

» Definition 7: Partial and Complete Binding

We refer to a pattern tp_i , which has no bound triple, as *unevaluated*, and write $*$ in the binding's b i^{th} position:

$$b = (t_1, \dots, t_{i-1}, *, t_{i+1}, \dots, t_n)$$

We call such a binding b *partial* and denote its evaluated patterns as $\mathcal{Q}(b) \subseteq \mathcal{Q}$, as well as its unevaluated patterns as $\mathcal{Q}^u(b) = \mathcal{Q} \setminus \mathcal{Q}(b)$. Binding b is *complete*, if all patterns have been evaluated: $\mathcal{Q}(b) = \mathcal{Q}$.

Binding b *comprises* a partial binding b' , if any matched triple t_i in b' is also contained in b at position i . In this case, we say b' *contributes* to b .

Query and Result Semantics. The semantics of hybrid queries follow those of SPARQL BGP queries. That is, a binding is a subgraph of the underlying data graph, which *matches* all query patterns.

» Definition 8: Structured Triple Pattern Match

Given a triple pattern $tp = \langle s, p, o \rangle$ in query \mathcal{Q} , where $s \in \mathcal{V}_V^Q \uplus \mathcal{V}_E$, $p \in \ell_a \uplus \ell_r \uplus \mathcal{V}_V^Q$, and $o \in \mathcal{V}_V^Q \uplus \mathcal{V}_C^Q$, and a triple $t = \langle s', p', o' \rangle$ in data graph \mathcal{G} , let $\mathbb{1}_M$ denote an indicator function such that:

$$\mathbb{1}_M(tp, t) := \begin{cases} (\text{IF}(s \in \mathcal{V}_V^Q) : \mathbf{T} \quad \text{ELSE} : s = s') \quad \text{AND} \\ \mathbf{T} \quad (\text{IF}(p \in \mathcal{V}_V^Q) : \mathbf{T} \quad \text{ELSE} : p = p') \quad \text{AND} \\ (\text{IF}(o \in \mathcal{V}_V^Q) : \mathbf{T} \quad \text{ELSE} : o = o') \\ \mathbf{F} \quad \text{ELSE} \end{cases}$$

Triple t matches the structured pattern tp iff $\mathbb{1}_M(tp, t) = \mathbf{T}$.

The only difference to SPARQL BGP queries is due to keyword nodes: a value node $o' \in \mathcal{V}_A$ matches a keyword $w \in \mathcal{V}_K^Q$, iff the bag-of-words from o' contains word w . We say keyword triple patterns have a *contains semantic*:

» **Definition 9: String Triple Pattern Match**

Given a keyword pattern $tp = \langle s, p, w \rangle$ in query \mathcal{Q} , where $s \in \mathcal{V}_V^Q \uplus \mathcal{V}_E$, $p \in \ell_a \uplus \mathcal{V}_V^Q$, and $w \in \mathcal{V}_K^Q$, and a triple $t = \langle s', p', o' \rangle$ in data graph \mathcal{G} , let $\mathbb{1}_M^K$ denote an indicator function such that:

$$\mathbb{1}_M^K(tp, t) := \begin{cases} (\text{IF}(s \in \mathcal{V}_V^Q) : \mathbf{T} \text{ ELSE} : s = s') \text{ AND} \\ \mathbf{T} \text{ (IF}(p \in \mathcal{V}_V^Q) : \mathbf{T} \text{ ELSE} : p = p') \text{ AND} \\ \text{(bag-of-words}(o') \cap \{w\} \neq \emptyset) \\ \mathbf{F} \text{ ELSE} \end{cases}$$

Triple t matches string pattern tp iff $\mathbb{1}_M^K(tp, t) = \mathbf{T}$.

2.2 QUERY PROCESSING

Next, we discuss a traditional pipeline for processing hybrid queries. We give an overview in Section 2.2.1 and outline query plan generation in Section 2.2.2. Further, we introduce query cost estimation basics (see Section 2.2.3) and present query execution in Section 2.2.4.

2.2.1 Overview

The first relational database was implemented in the System R project [149]. Nowadays, SPARQL and BGP query processing still follows the main steps introduced in [149]: (1) statistics generation, (2) query optimization, and (3) query execution, see [128]. A generic overview is shown in Figure 9-a.

① Statistics Generation

Statistics generation is done offline, i.e., before a query is issued. In this phase, attribute value, relation, and/or join cardinalities are computed. These cardinalities are oftentimes stored by means of histograms [135] or wavelets [118].

② Query Optimization

During the second step, a query is translated to an *execution plan* (query plan). The translation is done at runtime and specifies *how the query results are computed* [53]. This is possible because of the *declarative nature* of SPARQL queries: queries solely state which data shall be retrieved, but not how. More precisely, the translation is usually carried out in three stages, see Figure 9-b.

- Firstly, the query is parsed to a logical representation, e.g., a relational algebra [45, 142] and its syntax is validated [128].
- Secondly, a “logical” query plan is computed, e.g., via query unnesting [124] or view resolution/merging [72], that captures the query semantics [128].

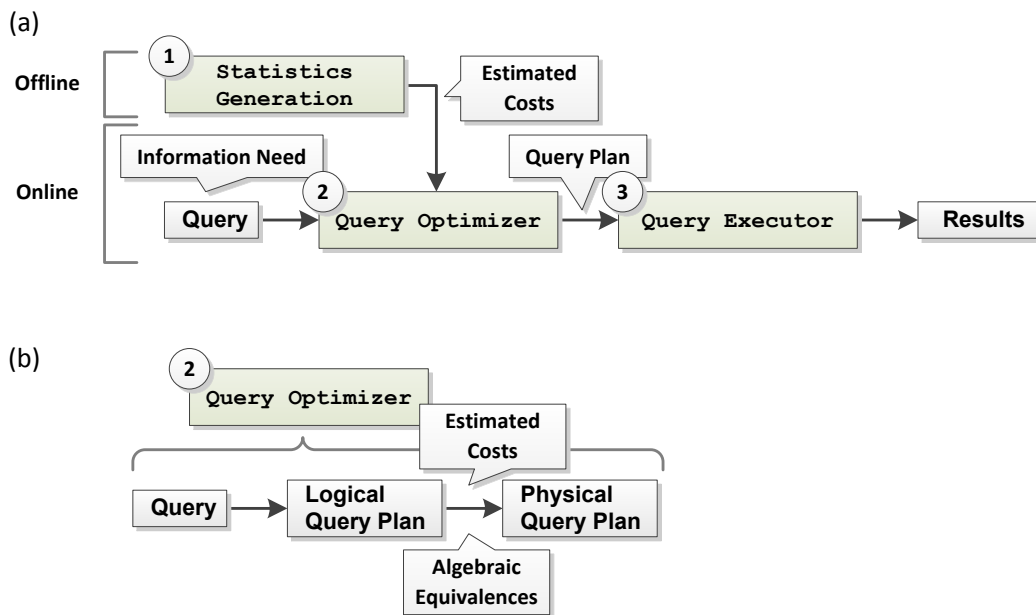


Figure 9: (a) Traditional query processing pipeline based on the System R project [149].
 (b) Query optimization producing a logical and physical query plan [128].

- Last, the logical query plan is transformed to a physical plan. The latter precisely specifies how query results will be computed, i.e., the plan states for each of its operators concrete algorithms (e.g., a hash join), and resources (e.g., disk or memory space) to be used. For this translation, a cost-based plan generation is frequently employed. That is, the optimizer tries to estimate computation costs of query subexpressions (e.g., single joins or triple patterns) and find a physical plan that minimizes its associated costs [128]. Notice that estimations may differ depending on the available statistics.

③ Query Execution

In the last stage, the physical query plan is given to an engine that computes the results – precisely as dictated by the physical plan [53]. Two aspects during query execution are important for this thesis: pipelining and scheduling.

- Pipelining is a desirable property of operators in the query plan. The property dictates that an operator processes one input at a time and propagates one output to the next operator. Pipelined computation may lead to better response times and higher throughput [53]. Unfortunately, not all operators (e.g., sorting) support pipelining, but instead must process all their (intermediate) results together [53].
- Scheduling of computation during the query plan execution is of high importance [53]. We will detail two options, namely iterator-based and data-driven scheduling, in Section 2.2.4.

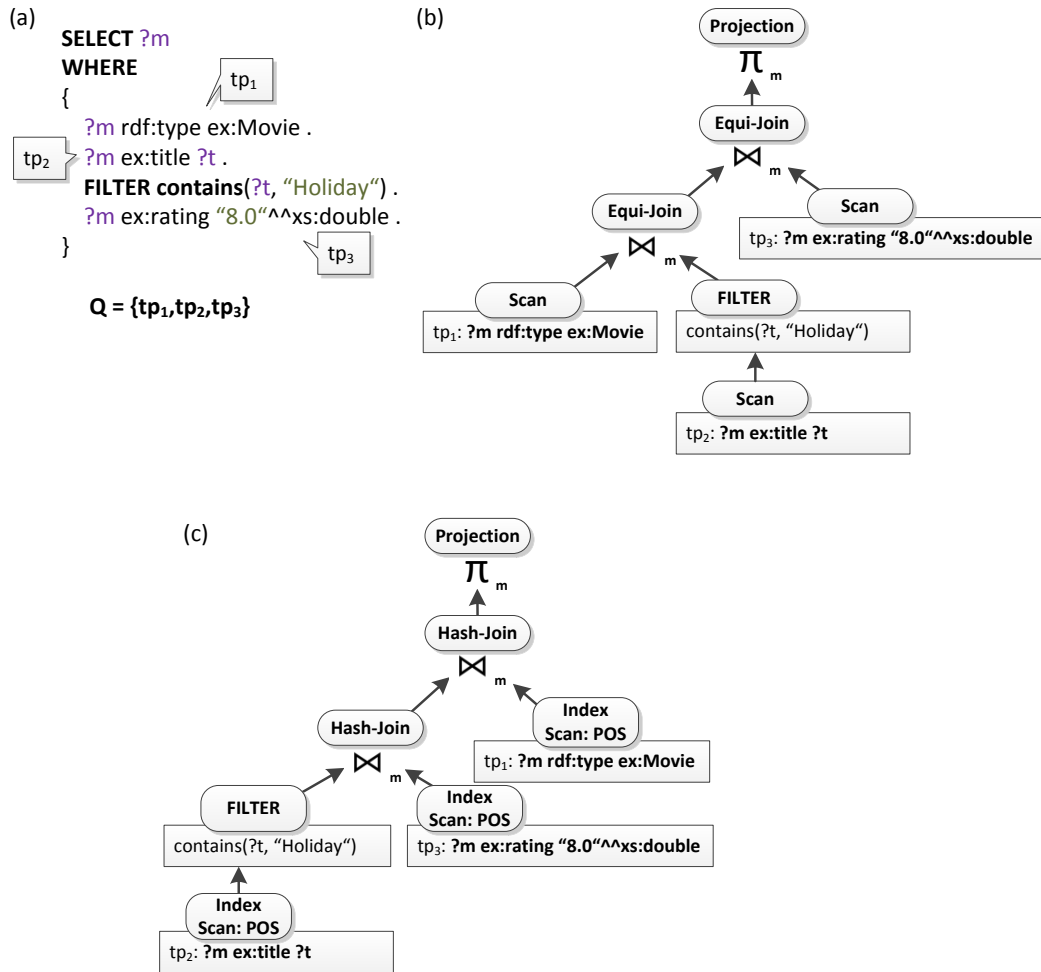


Figure 10: (a) Query fragment from example in Figure 8. (b) Logical query plan for the query in (a). (c) Physical query plan for the query in (a).

Example 10

Figure 10-a depicts a logical/physical query plan for a fragment of our running example from Figure 8.

The logical plan does not include any implementation specific aspects, e.g., concrete join algorithms. In contrast, the physical plan states that hash-joins shall be used. Further, the physical plan employs a different join ordering than the logical plan. That is, bindings for triple patterns are joined/materialized differently in the physical plan, because the patterns are sorted according to their estimated join cardinalities.

2.2.2 Query Optimization

As outlined above, query optimization transforms a query into a physical query plan. However, because of declarative query languages, such as SPARQL [17] or SQL [53], an optimization engine has a certain freedom to choose amongst multiple possible physical query plans, which all lead to the same result set. For-

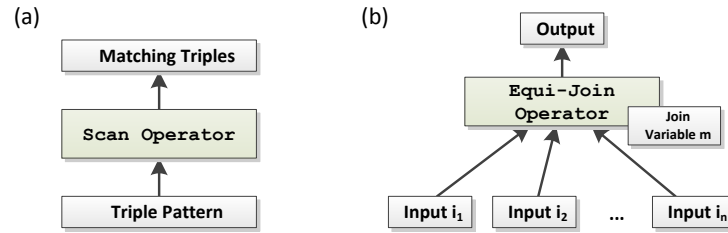


Figure 11: (a) Generic scan operator. (b) Generic equi-join operator with n join inputs and join variable m .

mally speaking, the logical representation of a particular query can be written in different forms, which are equivalent to each other (algebraic equivalences) [53].

These various plans may differ drastically in terms of their costs, e.g., number of intermediate results and overall query processing time. This problem is addressed by the query optimization engine – it aims at finding the “best” physical query plan for a given query. We will provide details on a cost model in Section 2.2.3.

Query optimization has a long-standing history in the (relational) database community. Initial work on query optimization was done in the System R project [149]. The authors in [149] proposed a bottom-up dynamic programming approach for join order optimization and introduced the notion “interesting orders”. Other solutions exploited rules for optimization. A well-known example is the Starbust optimizer [70] that employed rules in order to integrate low-level physical operators. In contrast to the bottom-up strategy proposed in [149], the works [63, 64] presented a top-down optimization relying on memorization techniques.


In this thesis, we have a have simplistic query model – we solely address BGP queries (see Definition 4, p. 20). In such a setting, previous approaches [104] focused on the choice of the query operators and the ordering of the joins. Note, there is a large body of works in the area of relational databases, which focus on more sophisticated optimizations [53]. Further, for general SPARQL queries additional optimizations have been investigated [108, 142, 161].

Query Operators. Two kinds of logical query operators are important for us: a scan and an equi-join operator. Intuitively, a scan operator materializes matching triples for a given pattern, see Figure 11-a.

» Definition 10: Scan Operator (Informal) [112]

For a given query Q over a data graph \mathcal{G} , a scan operator is a function that projects a triple pattern in Q to a set of matching triples in data graph \mathcal{G} .

There are a number of possible implementations (physical scan operators) for a logical scan operator [84, 139].

 **Example 11**

Given a vertical triple store, triples are stored directly in a three-column SPO-table and indexes are provided for every possible access pattern [84, 139]. In Figure 10-c, we use such a vertical store and implement the scan operators as scans over the POS-index:

P	O	→	S
ex:bornIn	ex:l1	→	ex:p2
ex:dateOfBirth	1976	→	ex:p1
...	...	→	...
ex:rating	8.0	→	ex:m1
...	...	→	...

For instance, the scan for pattern $tp = \langle ?m, \text{ex:rating}, 8.0 \rangle$ (see Figure 10-c) would be realized by means of a lookup with prefix $\langle \text{ex:rating}, 8.0 \rangle$ in the above POS-index.

Triples that match different triple patterns are combined via a logical equi-join operators, see Figure 11-b. Based on [112], we informally define an equi-join as:

» **Definition 11: Equi-Join Operator (Informal) [112]**

For a given query Q over a data graph \mathcal{G} , intermediate results from n inputs are combined by a theta-join via an equality constraint on the join variable m – the so-called equi-join on m . Each input i may either be another join or a scan operator.

Frequently, hash-joins, merge-joins, and nested-loop joins are exploited as physical join operators [84, 139]. For example, in Figure 10-c, the query optimizer decided to use two hash-joins. For simplicity we will discuss binary equi-joins in the following. However, all our approaches can be applied to n -ary joins.

Join Ordering. One of the key tasks of the optimizer is finding the “best” join order. That is, algebraic equivalences allow for multiple different join orderings, which all lead to the same results, but cause different costs in terms of number of intermediate results and query processing time. Most notably, it is known that the join operation is *commutative* and *associative* [53].

- Join commutativity: $tp_i \bowtie tp_j \Leftrightarrow tp_j \bowtie tp_i$.
- Join associativity: $tp_i \bowtie (tp_j \bowtie tp_k) \Leftrightarrow (tp_i \bowtie tp_j) \bowtie tp_k$.

Example 12

The join order in the logical query tree in Figure 10-b is changed to the order in the physical tree, see Figure 10-c. That is, $tp_1 \bowtie tp_2$ is evaluated first, instead of $tp_3 \bowtie tp_1$.

Notice, join (re-)ordering can result in different query tree forms [53]: a left-deep, a right-deep, or a bushy tree.

2.2.3 Cost Model

The question remains, how a query optimizer quantifies the quality of a physical query plan. For this, various approaches exist for SPARQL [104, 108, 142, 161] as well as SQL queries [53]. Intuitively, many of those approaches are (to some extent) cost-based, i.e., use optimization strategies to discover the cheapest physical plan over a search space.

Depending on the concrete system, a cost model may vary. However, these models oftentimes incorporate aspects as follows [53]:

- ① *Costs for Secondary Storage Access*
Costs associated with read and write access to data from the secondary disk storage, respectively.
- ② *Costs for Disk Storage*
Costs for storing data at disk during query processing.
- ③ *Costs for Computation*
Costs due to in-memory processing, e.g., join computations. Such costs are also referred to as CPU costs.
- ④ *Costs for Memory*
Costs associated with memory consumption for, e.g., buffering intermediate results.
- ⑤ *Costs for Communication*
In case of a distributed query processing setting, costs can be caused by communication of (intermediate) results to the site/terminal where the query was issued at.

For join ordering and query operator selection, selectivity estimation plays a crucial role [104]. In simple terms, selectivity estimation allows to compute cardinality estimations for a query and its sub-expressions. This way, many of the above described costs can be approximated for a particular physical query tree. More specifically, a selectivity estimation function is defined as:

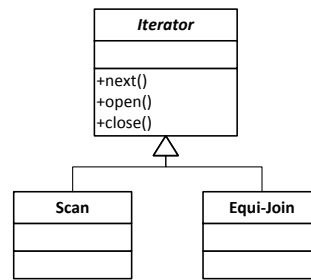


Figure 12: An abstract iterator class, implemented by a generic scan and equi-join operator.

» **Definition 12: Selectivity Estimation Function (Informal)**

Given a query Q and a data graph \mathcal{G} , a selection estimation function, sel , projects Q to the cardinality of query Q 's result set, which is computed over data graph \mathcal{G} .

Note, we will provide a refined definition in Definition 21.

2.2.4 Query Execution

In this last step of the query processing process (see Figure 9 for an overview of the query processing pipeline), the physical query plan is executed and query results are returned to the user. Two aspects are of importance during this phase [53]: pipelining capabilities of an operator and operator scheduling. The former can greatly improve the overall throughput of the query execution – we discuss pipelining in more details in Section 2.3.

The latter, the operator scheduling, dictates the order in which query operators interact with each other. Traditional database systems employ the *iterator principle* [62]. That is, each query operator implements the iterator interface (see Figure 12) – comprising three methods [62]:

- An open method that activates the operator and recursively its children.
- An iterator contains a next method, which produces the next binding. In particular, a join operator implementation of the next method would call the next method of its children and join their partial bindings.
- Last, an iterator features the close method, which terminates the operator.

As an alternative solution to the iterator principle, the *data-driven scheduling* has been proposed in [169]. Execution based on data-driven scheduling is sometimes called *push-based* query execution. Here, the data producing operators trigger all other operators and thereby control the overall query execution. Thus, “the data” and not “the operators” control the query execution. More precisely, every operator gets input data from its children and actively pushes outputs to its parent [169].

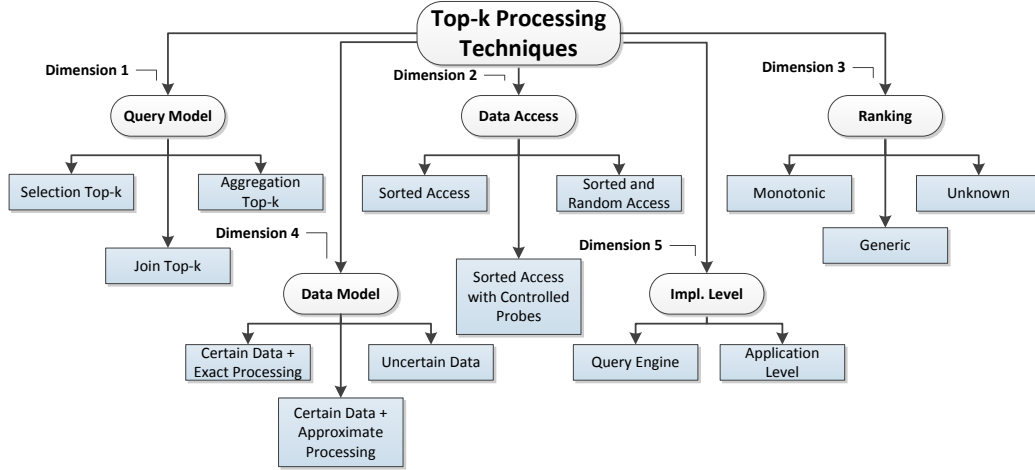
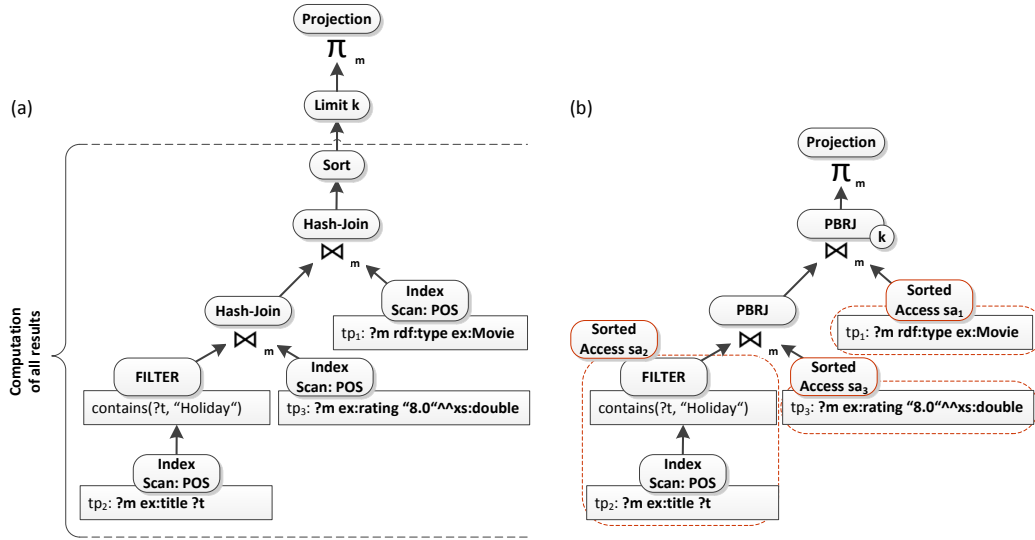


Figure 13: Overview of varying top-k strategies based on [95].

Figure 14: Given query Q in Figure 10-a: (a) Naive top-k query processing exploiting a sort operation after computing the entire result set. (b) Join top-k processing using two PBRJ operators and one sorted access for every triple pattern.

In the next chapters, we will rely on both query execution paradigms, depending on what precise setting we have. Most importantly, if and how data is distributed over a space of sources.

2.3 RANK-AWARE QUERY PROCESSING

As motivated in the introduction (see Section 1.1.2.3), top-k query processing aims at efficiently computing the k top-ranked results for a given query. For this, techniques try to *terminate early*, i.e. to not compute the entire result set, but stop computation immediately after the top- k results have been found [95].

In Section 2.3.1, we give a brief overview over existing top-k processing strategies. Further, we discuss a particular kind of top-k query processing in Section 2.3.2: join top-k processing.

2.3.1 Overview

In recent years, top-k processing techniques gained a significant amount of attention in the DB community [95]. While works differ along multiple dimensions (discussed below), they commonly target the so-called *early termination*.

Early Termination. Given a query Q , we can compute a top-k result simply by materializing the entire result set for Q and sorting the results afterwards, see Example 3. In other words, we would apply a sort operation after the top-level join in query tree.

Unfortunately, sorting is a blocking operation and does not allow for pipelining. In contrast, a top-k join operator (introduced in the next section) omits the sort operation and enables pipelining. This way, it is possible to stop result computation immediately after the first top-k results have been found [95] – commonly referred to as early termination.

Example 13

Figure 14 provides two physical query plans for the query in Figure 10-a. The LHS physical query plan computes all query bindings and sorts them in the very last step, since sorting is a blocking operation.

In contrast, the RHS physical query plan applies two top-k join operators. Here, a query optimizer can search for the best possible physical query plan – no blocking sort operation is applied. Moreover, the top-k join operators allow to terminate early. That is, not all results have to be computed, in order for the top-k results to be reported.

Design Dimensions. Based on [95], top-k techniques are categorized by means of dimensions as depicted in Figure 13:

- *Dimension 1: Query Model*
Approaches differ with regard to the query model. *Selection* top-k processing addresses entity queries only. That is, it computes top-ranked entities with every entity being ranked according one or more criteria, e.g., [54]. Furthermore, *join* top-k processing techniques rank every triple and calculate top-k join results, e.g., [93]. Last, the *aggregate* top-k processing focuses on aggregate queries, e.g., [110].
- *Dimension 2: Data Access*
Top-k processing differs depending on the available data access. Frequently, one assumes that data, e.g., triples or entities, can be accessed in descending score order (*sorted access*, defined below), e.g., [54, 66]. In addition, other approaches rely on a *random access* to their input data, e.g., [54, 65]. Some works, e.g., [93, 32], require at least one sorted access, in order to use the random accesses for probing the remaining inputs.
- *Dimension 3: Ranking*
Top-k strategies vary depending on the ranking functions they support. While most approaches require ranking functions to be monotonic, e.g.,

[54], some other works lifted this restriction [174]. Last, some approaches that target skyline queries require no ranking function, e.g., [28, 172].

- *Dimension 4: Data and Query Uncertainty*

Data model and query model could be uncertain. Former approaches deal with uncertain data [137, 152], e.g., in the sensor networks domain, where sensor measurements are never exact and always include some amount of jitter. The latter approaches address approximate query processing over certain data. Here, the goal is to trade off result accuracy in favor of result computation time [14, 157].

- *Dimension 5: Implementation Level*

Top-k techniques can vary with regard to their implementation. Some approaches are realized in the application layer, on top of a query processing system, e.g., [34, 50]. Other solutions, however, are implemented as part of the query processing engine, i.e., as query operators, e.g., [93, 109, 110].

The interested reader may see [95] for a comprehensive survey discussing above dimensions and top-k strategies in depth.

In this work, we are concerned with *join top-k* (Dimension 1) over *certain data* (Dimension 4), implemented as *query operators* (Dimension 5), and requiring a *sorted access* (Dimension 2) as well as *monotonic* ranking functions (Dimension 3). We will define the precise setting in Chapter 3 and Chapter 5.

2.3.2 Top-k Join Processing

Queries over Web data often comprise joins to combine bindings for multiple triple patterns. For example, the rather simple query in Figure 8 would already require seven joins. We therefore focus on the *top-k join* (Dimension 1) problem in this thesis. The top-k join commonly requires a sorted access as well as a monotonic ranking function.

2.3.2.1 Ranking Function

We employ a *ranking function* for quantifying the relevance of a binding b :

» **Definition 13: Ranking Function**

A *ranking function* is given by $\text{score}_Q : \mathbf{B} \mapsto \mathbb{R}$, with \mathbf{B} as set of all bindings for query Q . $\text{score}_Q(b)$ assigns a score to b , which indicates b 's relevance with regard to query Q and/or the user, who issued query Q .

More precisely, $\text{score}_Q(b)$ is given by an aggregation over b 's triples: $\text{score}_Q(b) := \bigoplus_{t \in b} \text{score}_Q(\vec{t})$, with \bigoplus as *monotonic aggregation function*.

The above definition follows the notion of user-/query-dependent ranking as presented in [12, 36, 156]. A prime example for query-dependent ranking functions is result ranking for keyword queries, where scores reflect the quality of the keyword match. This quality could, e.g., be measured by the Levenshtein distance [26].

On the other end of the spectrum are “offline” ranking functions, which assign scores that are independent of query and user characteristics, respectively. Such triple scores could, e.g., be obtained via PageRank inspired ranking [74] or witness counts [52].

Note, score_Q could be defined *as part of the query*, e.g., by means of the ORDER BY clause in SPARQL.

Example 14

Given the query in Figure 8 and a binding

$$b = (t_1 = \langle s_1, p_1, o_1 \rangle, \dots, t_8 = \langle s_8, p_8, o_8 \rangle)$$

where triple t_i matches triple pattern tp_i . Then, a ranking function for b could be defined as:

$$\begin{aligned} \text{score}_Q &= \text{pageRank}(s_1) \\ &+ (1 - \text{levenshtein}(o_2, \text{“Holiday”})) \\ &+ \text{pageRank}(s_4) \\ &+ (1 - \text{levenshtein}(o_4, \text{“Audrey”})) \\ &+ (1 - \text{levenshtein}(o_7, \text{“Belgium”})) \end{aligned}$$

where $\text{levenshtein}()$ is a function measuring the Levenshtein distance [26] between an object (e.g., o_2) and a keyword (e.g., “Holiday”). Further, the PageRank [132] score of an entity is captured by $\text{pageRank}()$ and the aggregation function, \oplus , is given by a summation.

2.3.2.2 Sorted Access

As a special scan operator (see Definition 10 and Figure 14), we require a *sorted access* sa_i for every pattern tp_i in query Q , which retrieves *matching triples in descending score order*. Formally, a sorted access is defined as:

» Definition 14: Sorted Access

Given a data graph \mathcal{G} , a query Q , and a ranking function score_Q , a *sorted access* sa_i for a pattern tp_i in Q is a scan operator that projects tp_i to a *sorted set* of matching triples in graph \mathcal{G} . That is, each triple is assigned a score via score_Q and the set is sorted according to descending score order.

Efficient sorted access implementations for RDF data have been proposed in [2, 115]. Let us illustrate the core idea on our running example:

Example 15

Let us continue Example 14 and provide sorted accesses for its ranking function.

- On one hand, given the keyword pattern

$$tp_2 = \langle m, \text{title}, \text{"Holiday"} \rangle$$

in Figure 8, a sorted access must materialize all triples, which have an attribute node that contains “Holiday”. After materialization at runtime, these triples are sorted with descending similarity with regard to that keyword (measured via the Levenshtein distance). Thus, sorted access sa_2 loads one triple

$$t = \langle m_1, \text{title}, \text{"Roman Holiday"} \rangle$$

which comprises “Holiday”. Then, triple t is ranked according to its Levenshtein distance to “Holiday”. Similarly, sorted accesses for patterns tp_4 and tp_7 (see Figure 8) can be provided.

- On the other hand, for pattern

$$tp_6 = \langle p, \text{type}, \text{Person} \rangle$$

in Figure 8, an offline ranking based on a PageRank score for Person instances is employed, see Example 14. So, we can index all triples matching pattern tp_4 based on their associated Person score at offline time. Finally, a sorted access can be provided at runtime by iterating over matching triples in descending PageRank score order. Similarly, sorted accesses for patterns tp_1 , tp_3 , tp_5 , and tp_8 (see Figure 8) can be provided.

Partial bindings retrieved from sorted accesses are combined via equi-joins, see Definition 11 and Figure 14 – as presented in Section 2.2. To enable a top-k query processing, we introduce a *rank-aware equi-join* in the following section.

2.3.2.3 Pull Bound Rank Join Framework

The authors in [143] formulated a general framework for top-k join algorithms, the Pull Bound Rank Join (PBRJ), which can be instantiated to yield well-known top-k join approaches such as the hash rank-join (HRJN) or the nested-loops rank-join (NRJN) [93].

This framework contains two components: a pulling strategy \mathcal{PS} and a bounding strategy \mathcal{BS} . The pulling strategy decides, which join input to pull next. The bounding strategy estimates a maximal possible score for “unseen” join results, i.e., the maximal score of future join results.

Notice, for simplicity we outline the framework for binary joins, however, it can be easily extended to cover n-ary joins.

Algorithmus 1 : Pull/Bound Rank Join framework [143].

Params : Pulling strategy \mathcal{PS} and bounding strategy \mathcal{BS}
Index : Sorted access sa_i and sa_j for input i and j , respectively.
Buffer : Output buffer \mathbf{O} . Buffer \mathbf{H}_i and \mathbf{H}_j for “seen” bindings, pulled from sa_i and sa_j .
Input : Query \mathcal{Q} and result size k .
Output : Top- k result in buffer \mathbf{O} .

```

1 begin
2    $\beta \leftarrow \infty$ 
3   while  $|\mathbf{O}| < k$  or  $\min_{b' \in \mathbf{O}} \text{score}_{\mathcal{Q}}(b') < \beta$  do
4     // select next input  $i$  to pull from
4      $i \leftarrow \mathcal{PS}.\text{input}()$ 
5     // pull next binding from  $i$ 
5      $b \leftarrow \text{next triple pattern binding from } sa_i$ 
6     // update upper bound  $\beta$ 
6      $\beta \leftarrow \mathcal{BS}.\text{update}(b)$ 
7     // join attempt with seen bindings from input  $j$ 
7      $\mathbf{O} \leftarrow \mathbf{H}_j \bowtie \{b\}$ 
8     Retain only  $k$  top-ranked bindings in  $\mathbf{O}$ 
9     // update seen buffer  $\mathbf{H}_i$ 
9     Add  $b$  to  $\mathbf{H}_i$ 
10    // return top- $k$  results
10    return  $\mathbf{O}$ 

```

Framework. The PBRJ framework [143] is depicted in Algorithm 1. On Line 3, we check whether the output buffer \mathbf{O} comprises k complete bindings and if there could be unseen (future) bindings with higher scores – measured via bound β and bounding strategy \mathcal{BS} , respectively. If both conditions hold, the PBRJ terminates and reports bindings in \mathbf{O} (*early termination*, as explained above). Otherwise, \mathcal{PS} selects an input i to pull from (see Line 4) and produces a new partial binding b from the sorted access on input i , see Line 5. After materialization, we update β using the bounding strategy \mathcal{BS} . Then, we attempt to join binding b with seen bindings from the other input, j , and add join results to output buffer \mathbf{O} , see Lines 7-8. Last, we put partial binding b in buffer \mathbf{H}_i (Line 9).

Instantiations for \mathcal{PS} and \mathcal{BS} . Multiple works proposed bounding strategies, e.g., [55, 93, 116, 143] and pulling strategies, e.g., [93, 117]. Commonly, the *corner-bound* [93] is employed as bounding strategy:

» Definition 15: Corner Bound

A PBRJ operator maintains two bounds, u_i and l_i , for each input i . u_i is the highest score observed from input i , while l_i is the lowest observed score on input i . If input i is exhausted, l_i is set to $-\infty$. The bound for scores of unseen join results is:

$$\beta := \max\{u_1 \oplus l_2, u_2 \oplus l_1\}$$

where \oplus is the aggregation function used in the ranking function (see Definition 13, p. 32).

On the other hand, the *corner-bound-adaptive strategy* [93] is frequently used as pulling strategy \mathcal{PS} :

» Definition 16: Corner-Bound-Adaptive Pulling

The corner-bound-adaptive pulling strategy chooses the input i such that: $i = 1$ if $u_1 \oplus l_2 < u_2 \oplus l_1$ and $i = 2$ otherwise, where \oplus is the aggregation function used in the ranking function.

If $u_1 \oplus l_2 = u_2 \oplus l_1$, the input with the smaller number of unseen (future) partial bindings is chosen.

Note, the corner-bound has been shown to be instance-optimal for binary PBRJ operators and one ranking attribute per join input [144].

RANK-AWARE QUERY PROCESSING

RANK-AWARE QUERY PROCESSING

Context of this Chapter. In this chapter, we present an exact top-k join for distributed RDF data: *Linked Data top-k processing*. This approach is based on our previous publication [2] and resembles a rank-aware equi-join operator for distributed Web data.

More specifically, our Linked Data top-k processing allows a data-driven rank-aware processing of hybrid queries over distributed Web data. Data-driven scheduling of operators is a key feature of our work and enables the processing to be less affected by Web data source unavailabilities. In fact, processing is driven by data retrieved from Web sources. Moreover, we leverage information, which is available in our Web data indexes, in order to significantly improve the corner bound strategy and to allow for an earlier termination. This tighter bound drastically saves time and number of triples/Web data sources processed during top-k result computation.

These contributions aim at Research Question 1: How to enable top-k query processing on highly distributed Web data? As a complementary approach, which targets Research Question 4, we will present an approximate top-k processing for Web data in Chapter 5. In particular, this work will allow for efficiency gains by means of less accurate result sets. In contrast, the Linked Data top-k approach processes queries in an exact and complete manner.

We classify our approach based on the design dimensions outlined in Section 2.3.1. This classification is highlighted in Figure 15.

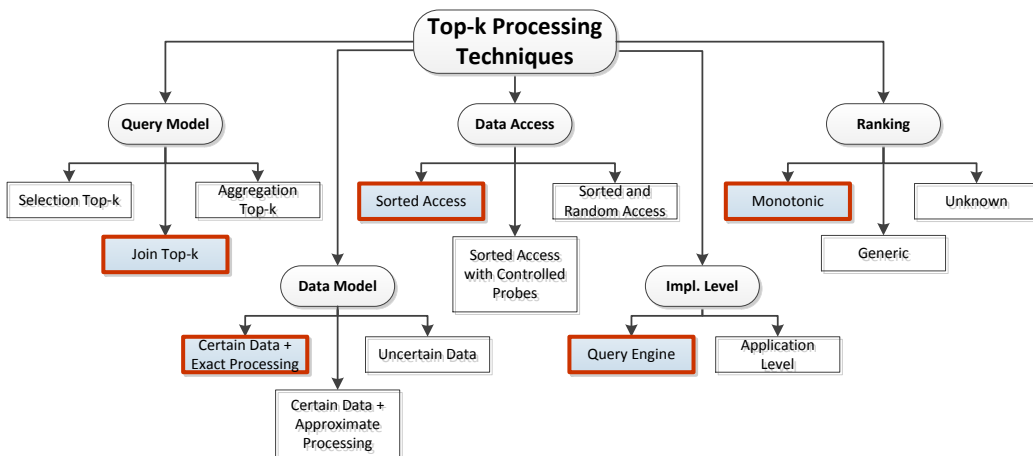


Figure 15: Classification of our *Linked Data top-k processing* approach.

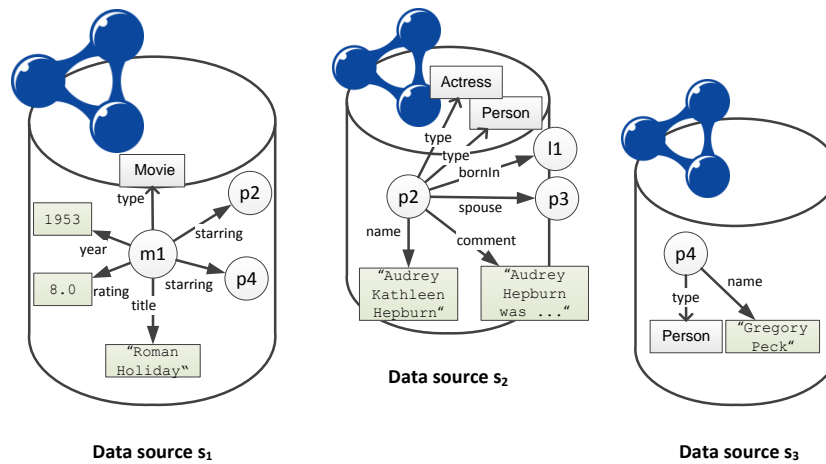


Figure 16: The running example from Figure 7 (adapted for simplicity) is distributed over three data sources: s_1 , s_2 , and s_3 . The RDF data describes the movie “Roman Holiday” and its actors Audrey Hepburn and Gregory Peck.

- We target a top-k join problem, i.e., single triples have ranking scores assigned and top-ranked joined results are reported.
- Further, we have certain data and compute exact results.
- We provide a novel lightweight sorted access implementation for Web data sources. In particular, the necessary indexes make maintenance of changing Web data highly efficient, due to our simple score statistics.
- We implemented a top-k join operator, which can be integrated in a physical query plan – as decided by a query optimizer. In fact, we present a selectivity estimation approach in Chapter 4, which allows for highly accurate selectivity estimations of hybrid queries over Web data. Based on these selectivity estimates, a query optimizer can create a physical query plan – including our Linked Data top-k join operator.
- Last, our approach employs a monotonic ranking function. Note, many common ranking functions, e.g., PageRank ranking [132], fall in this category.

Outline. In Section 3.1, we introduce and motivate the problem (Section 3.1.1) as well as provide the necessary background on Linked Data query processing (Section 3.1.2). Then, we outline our contributions and research questions in Section 3.2. In Section 3.3, we introduce our novel top-k processing approach. In particular, we propose two optimizations allowing for tighter score bounds on future join results, and a way to prune unnecessary partial bindings. Last, we present our evaluation in Section 3.4 and discuss related work in Section 3.5, before concluding with Section 3.6.

3.1 INTRODUCTION

3.1.1 Motivation

RDF data is oftentimes highly *distributed* across a space of sources. Each data source can comprise one or more RDF graphs. Formally speaking, a data graph \mathcal{G} could be a disconnected graph, where every data source holds one or more subgraphs of \mathcal{G} . Notice, this general notion of distributed data is strongly related to the concept of *dataspaces* – the interested reader may see [56, 71].

 **Example 16**

The data from our running example in Figure 7 is distributed over three data sources (s_1 , s_2 , and s_3) in Figure 16.

One popular form of distributed RDF is *Linked Data* [24]. The Linked Data principles²⁷ mandate how to access and publish RDF data on the Web:

- ① Use Uniform Resource Identifiers (URIs) as identifiers for entity nodes \mathcal{V}_E in the data graph \mathcal{G} .
- ② Use Hypertext Transfer Protocol (HTTP) URIs to enable an easy lookup.
- ③ Dereferencing a URI returns a description, i.e., RDF graph, of the entity identified by that URI.
- ④ Entity descriptions should link to related entities, which are again identified via their HTTP URIs.

According to the Linked Data principles, dereferencing a Linked Data URI via HTTP should return a machine-readable description of the entity identified by that URI. So, each entity in \mathcal{V}_E represents a data source:

» **Definition 17: Linked Data Source**

Given a data graph \mathcal{G} , its subgraphs are distributed over a space of sources: $\{s_i\}$. A HTTP URI d is an identifier for a Linked Data source s , which features a set of RDF triples \mathcal{T}_s^d from \mathcal{G} as content. That is, \mathcal{T}_s^d is obtained by dereferencing d . Triples in \mathcal{T}_s^d contain HTTP URI references (links) that connect d with related sources.

 **Example 17**

Figure 17 depicts three Linked Data sources (s_1 , s_2 , and s_3) identified by means of their URIs: `ex:m1`, `ex:p2`, and `ex:p4`. Dereferencing their URIs would yield the RDF graphs that are shown in Figure 17.

²⁷<http://www.w3.org/DesignIssues/LinkedData.html>, retrieved 2013-11-15.

Listing 2: Namespace prefixes.

```

1 @prefix ex: <http://example.org/> .
2 @prefix xs: <http://www.w3.org/2001/XMLSchema#> .
3 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

```

Listing 3: Source s_1 for URI $ex:m1$.

```

1 ex:m1 ex:year "1953"^^xs:int ;
2       ex:rating "8.0"^^xs:double ;
3       rdf:type ex:Movie ;
4       ex:title "Roman Holiday" ;
5       ex:starring ex:p2 ,
6                   ex:p4 .

```

Listing 4: Source s_3 for URI $ex:p4$.

```

1 ex:p4 ex:name "Gregory Peck" ;
2       rdf:type ex:Person .

```

Listing 5: Source s_2 for URI $ex:p2$.

```

1 ex:p2 ex:name "Audrey Kathleen Hepburn" ;
2       ex:comment "Audrey Hepburn was
3                   a British actress and humanitarian.
4                   Born in Ixelles, Belgium as
5                   Audrey Kathleen Ruston" ;
6       rdf:type ex:Person ;
7       rdf:type ex:Actress ;
8       ex:spouse ex:p3 ;
9       ex:bornIn ex:l1 .

```

Figure 17: The sources from Figure 16 are illustrated as Linked Data sources. That is, each source is identified with an URI and sources provide links to each other. RDF data is written in NTriples syntax [21].

Several works have studied the problem of Linked Data query processing [75, 77, 78, 105, 106, 140]. In fact, a recent survey provides an overview of Linked Data processing strategies [76]. Processing structured queries over Linked Data can be seen as a special case of federated query processing.

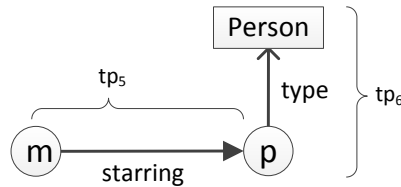
However, instead of relying on endpoints that provide structured querying capabilities (e.g., SPARQL interfaces), only HTTP URI lookups are available. Thus, entire sources have to be retrieved. Even for a single trivial query, hundreds of sources have to be processed in their entirety [105]. Aiming at delivering up-to-date results, sources often cannot be cached, but have to be fetched from external hosts. Thus, query processing efficiency/scalability are essential problems in the Linked Data setting.

In the next paragraphs, let us briefly discuss how existing query processing approaches addressed these efficiency and scalability issues.

```

1 PREFIX ex: <http://example.org/>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3
4 SELECT ?m ?p
5 WHERE
6 {
7   ?m ex:starring ?p .
8   ?p rdf:type ex:Person .
9 }

```

**Ranking Function:**

$$\text{score}_Q(b_1) = \text{popularity}(m_1) + \text{popularity}(p_1)$$

with binding $b_1 = \langle m_1, \text{starring}, p_1 \rangle, \langle p_1, \text{type}, \text{Person} \rangle$

Figure 18: Example query asking for movies, which star some Person. The query comprises two triple patterns, tp_5 (Line 7) and tp_6 (Line 8), and represents a fragment of the query in Figure 8. For ranking results, a popularity-based score is employed for movie as well as actor bindings.

3.1.2 Data-driven Linked Data Query Processing

In this chapter, we use the example query in Figure 18, which is a fragment of our running example in Figure 8. We use this query (as well as its matching data sources in Figure 17) to illustrate Linked Data query processing.

Example 18

For the query in Figure 18, the URIs `ex:m1`, `ex:p2`, and `ex:p4` (data is illustrated in Figure 17) are dereferenced and their triples are joined to produce bindings for the variables `m` and `p`.

The results are retrieved from different sources, which vary in relevance: we use an offline computed popularity score for bindings to the movie as well as the actress variable. More specifically, URI `ex:p2` provides data about the very popular actress Audrey Kathleen Hepburn, while URI `ex:p4` holds data about Gregory Peck, who is less well-known. Such differences in actor popularity could be captured by a ranking function, which aggregates the actor/movie popularity.^a

The ranking function score_Q is given as simple summation over those popularity scores:

$$\text{score}_Q(b) = \text{popularity}(m_1) + \text{popularity}(p_1)$$

where $b = (\langle m_1, \text{starring}, p_1 \rangle, \langle p_1, \text{type}, \text{Person} \rangle)$.

^aIn a real-world application, popularity scores could be computed based on a PageRank strategy [132], which exploits the ranking of the corresponding DBpedia and Wikipedia page, respectively.

Linked Data Query Processing. Traditionally, a query Q is evaluated by obtaining bindings for each triple pattern and then performing a series of equi-joins between bindings obtained for patterns that share a variable, see Section 2.2. In the Linked Data context, BGP queries are evaluated against all sources in the Linked Data graph \mathcal{G} . While some sources may be available locally, others have to be *retrieved via HTTP dereferencing during query processing*.

For this, exploration-based *link traversal* [77, 78] can be performed *at runtime*. The link traversal strategy assumes that Q contains at least one URI d as “entry point” to data graph \mathcal{G} . Starting from triples in \mathcal{T}_s^d , \mathcal{G} is then searched for results by following links from d to other sources.

Instead of exploring sources at runtime, knowledge about (previously processed) Linked Data sources in the form of statistics has been exploited to determine and rank relevant sources [75, 105] *at query compilation time*. Existing approaches assume a *source index*, which is basically a map that associates a triple pattern tp with sources containing triples that match tp . Let the result of a lookup in the source index for tp be $\text{source}(tp)$.

Given a source index, Linked data query processing can be conceived as a series of operators:

- *Source-Scan Operator*
We identify the *source-scan* as a distinctive operator in Linked Data query processing. Given a source s with URI d , $\text{source-scan}(d)$ outputs all triples in \mathcal{T}_s^d .
- *Selection Operator*
A *selection* $\sigma_{\mathcal{T}_s^d}(tp)$ is performed on \mathcal{T}_s^d to output triples that match a triple pattern tp .
- *Equi-Join Operator*
Two triple patterns tp_i and tp_j that share a common variable are combined via an equi-join operator $tp_i \bowtie tp_j$ (i.e., bindings for tp_i and tp_j are joined). In general, $Q_i \bowtie Q_j$ joins any subexpression Q_i with another subexpression Q_j , where $Q_i \subset Q$, $Q_j \subset Q$, and $Q_i \cap Q_j = \emptyset$. Note, in the following, we refer to an equi-join simply as *join*.
- *Union Operator*
Last, we have $\bigcup(i_1, \dots, i_n)$, which outputs the *union* of its inputs i .

For clarity of presentation, we assume triple patterns form a connected graph such that a join is the only operator used to combine triples from different pat-

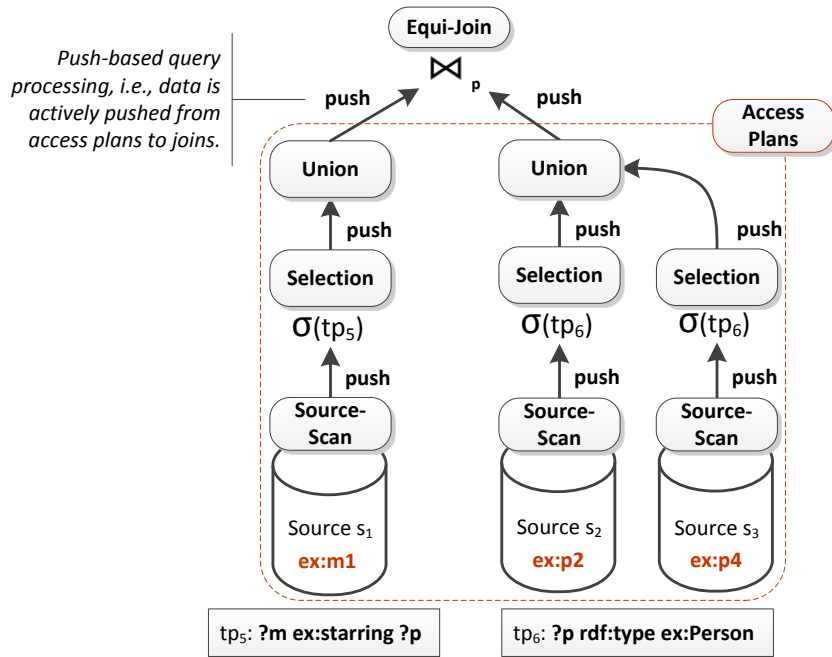


Figure 19: Push-based plan for query Q in Figure 18 [105, 106].

terns. Then, Linked Data query processing can be modeled as a tree-structured plan as exemplified (see Example 19).

Query plans in relational databases generally consist of access plans for individual relations. Similarly, Linked Data query plans are composed of *access plans* at the bottom-level – one access plan for each triple pattern. An access plan for triple pattern tp_i in query $Q = \{tp_1, \dots, tp_n\}$ is a tree-structured query plan constructed in the following way:

- ① At the lowest level, leaf nodes are source scan operators, one for every source s that is relevant for triple pattern tp_i , i.e., s has URI $d \in \text{source}(tp_i)$.
- ② The next level contains selection operators. We employ one selection operator for every source scan operator.
- ③ The root node is a union operator $\bigcup(\sigma_{\mathcal{T}d_1}(tp_i), \dots, \sigma_{\mathcal{T}d_n}(tp_i))$, which combines the outputs of all selection operators for tp_i (with $d_i \in \text{source}(tp_i)$).

Note, an access plan for triple pattern tp_i implements the scan operator defined in Definition 10, i.e., it provides access to matching triple for a given pattern tp_i in query Q .

At the next levels, the outputs of the access plans (of their root operators) are successively joined to process all triple patterns of the query, resulting in a *tree of operators*.

Example 19

Figure 19 shows an example query plan for the query in Figure 18. There are three source scan operators (one for each source): `source-scan(ex:m1)`, `source-scan(ex:p2)`, and `source-scan(ex:p4)`.

Together with selection and union operators, they form two access plans for the patterns tp_1 and tp_2 . The output of these access plans is combined using one join operator.

Push-based Processing. In previous works [105, 106], data-driven (also known as *push-based*) execution using symmetric hash join operators was shown to have better performance than iterator-based implementations (such as [78]). In a push-based model, operators push their results to subsequent operators instead of pulling from input operators, i.e., the execution is driven by the incoming data. This leads to better behavior in network settings, because, unlike in iterator-based execution models, the query execution is *not blocked*, when a single source is delayed [106]. See also Figure 19 for a push-based query plan.

3.1.3 Problem

Above query processing techniques are not well-suited for queries with ranked results such as Example 18. This is because sorting (of the ranked results) is a blocking operation. Thus, all results must be computed – even if the user is only interested in few top-ranked bindings. This *blocking nullifies the advantages of push-based query processing*, which aimed at allowing a non-blocking query plan. The problem is exacerbated by the fact that result computation on Linked Data requires retrieval of entire sources. So, data materialization from remote Linked Data sources may be much more expensive than in a centralized query processing context.

Top-k query processing – as outlined in Section 2.3 – aims at a more efficient query execution by focusing on the k best results, while skipping the computation of remaining results. This early termination can lead to a significant reduction in the number of inputs to be read and processed, which translates to drastic performance improvements.

Unfortunately, traditional top-k processing strategies are not suitable for Linked Data query processing: First, existing works require heavyweight indexes, which would cause extensive maintenance efforts given the frequently changing Web data. Second, previous top-k processing assumes an iterator-based query execution. However, as shown in [105, 106], push-based query execution is much more efficient for Linked Data query processing.

In the following, we show how to overcome these obstacles, by introducing the first join top-k strategy, which is well-suited for Linked Data query processing.

3.2 RESEARCH QUESTIONS AND CONTRIBUTIONS

Let us outline the research questions, hypotheses, and contributions, which we target throughout the chapter.

3.2.1 *Research Questions and Hypotheses*

As presented in Section 1.3, our overall research question is: How to allow for rank-aware and approximate query processing on the Web of data? In this chapter, we address the former part, i.e., we introduce a rank-aware equi-join operator for Web data. More specifically, we aim at Research Question 1:

🔗 **Research Question 1**

How to enable top-k query processing on highly distributed, schemaless Web data?

For addressing above research question, we verify hypotheses as follows:

□ **Hypothesis 1**

Join top-k processing based on the Pull Bound Rank Join (PBRJ) framework (see Algorithm 1) can be extended to match the requirements of highly distributed Web data. Moreover, such a top-k processing allows for significant performance gains for computation of ranked results over Web data.

Intuitively, Hypothesis 1 states that the PBRJ framework can be extended for the needs of Web data. In particular, we expect that the PBRJ can be extended to a push-based query processing – thereby omitting any blocking in the query execution due to Web source delays. Further, we expect that we can implement a lightweight sorted access over the Web data sources, which allows for an efficient maintenance.

To validate Hypothesis 1 we provide Algorithm 2, Algorithm 3, and Algorithm 4 in Section 3.3 – our *LD-PBRJ framework*. Moreover, we implemented these algorithms and empirically show (see the evaluation in Section 3.4) the feasibility of our sorted access as well as LD-PBRJ framework.

Further, due to the early termination feature of the LD-PBRJ framework, we expect to save computation time as well as process less join inputs. We empirically validate this claim by means of the evaluation in Section 3.4.

□ **Hypothesis 2**

Given our lightweight sorted accesses, we can improve the state-of-the-art bounding strategy (corner bound, see Definition 15, p. 36). Moreover, the LD-PBRJ framework can be extended to allow for pruning of partial bindings, which cannot lead to a complete top-k binding. This way, we process less partial bindings and save computation time due to less join attempts.

We provide a theoretical analysis by means of Theorem 1, Theorem 2, and Theorem 3 to validate both claims in Hypothesis 2. Furthermore, we show the validity of Hypothesis 2 in our evaluation in Section 3.4.

3.2.2 Contributions

While being naturally appealing, top-k processing has not been studied in the Linked Data context before. Aiming at above hypotheses, we provide the following contributions:

- *Contribution for Hypothesis 1*

Top-k query processing has been extensively studied for relational data [95]. Closest to our work is top-k querying over Web-accessible databases [170] and distributed join top-k processing [51].

However, the Linked Data context is unique to the extent that only URI lookups are available for accessing data. Instead of retrieving partial bindings from sources that are exposed via query interfaces (of the corresponding database endpoints), we have to *retrieve entire sources via URI lookups*.

These unique Web data characteristics require a novel top-k query processing strategy. In particular, the highly distributed Web data sources make a non-blocking push-based query execution essential – as outlined in the motivation section. Additionally, the frequently changing Web data requires a more lightweight sorted access implementation. Existing works did not face this issue, because data was oftentimes much more static and maintained at few/large databases.

Facing these characteristics, we propose the LD-PBRJ framework. To the best of our knowledge, this is the first work towards top-k Linked Data query processing.

- *Contribution for Hypothesis 2*

Based on our sorted access indexes (implementation), we provide a more accurate bounding strategy. In particular, we formally show that: (1) our bound is correct in Theorem 1, and (2) our bound is tighter than the current state-of-the-art strategy in Theorem 2.

Further, we propose an aggressive pruning of partial bindings that cannot contribute to the final top-k result. We formally show in Theorem 3 that this pruning still guarantees exact and complete top-k results.

Both optimizations lead to less join inputs read/processed and an earlier termination of the LD-PBRJ operator. We empirically show these performance gains in our evaluation.

- *Contribution for Hypothesis 1 and Hypothesis 2*

We conducted an evaluation on real-world Linked Data sources and queries to validate the Hypotheses 1 and 2. In these experiments the LD-PBRJ framework could achieve significant performance gains over the state-of-the-art Linked data query processing.

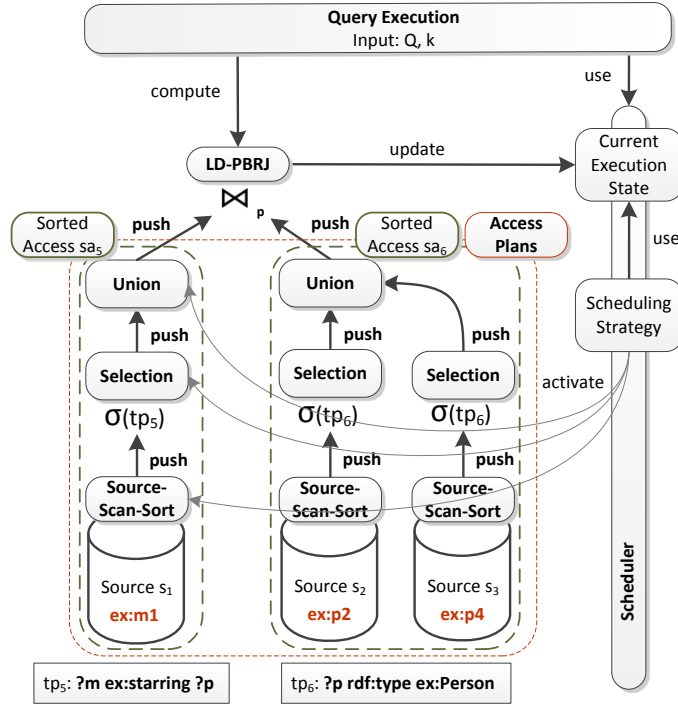


Figure 20: Rank-aware query plan for query Q in Figure 18 employing source-scan-sort operators, the LD-PBRJ in Algorithm 2, as well as a scheduler as replacement for a pulling strategy.

In fact, we can show that LD-PBRJ processing leads to 35% less computation time on average. We can further show that our proposed optimizations increase the computation time performance by 12% on average.

3.3 LINKED DATA TOP-K QUERY PROCESSING

We now discuss how existing *top-k join* (also called *rank join*) strategies can be extended to the Linked Data query processing problem as presented before. Further, we present an optimization towards tighter bounds and an aggressive result pruning. Throughout the query processing we do not approximate. Thus, our approach always reports correct and complete top-k results.

3.3.1 Sorted Access

Besides the source index employed for Linked Data query processing, we need a *ranking function* as well as a *sorted access* for top-k processing.

Regarding the former, we employ a monotonic ranking function, $score_Q$, for determining the relevance of triples and (partial) query results. That is, any function adhering to Definition 13 may be used.

Example 20

Continuing Example 18, the ranking function is:

$$\text{score}_Q(b) = \text{popularity}(m_1) + \text{popularity}(p_1)$$

where $b = (\langle m_1, \text{starring}, p_1 \rangle, \langle p_1, \text{type}, \text{Person} \rangle)$.

For the sake of simplicity, we assume that $\text{score}_Q(b)$ assigns triples in source s_1 (URI `ex:m1`) a score of 1, source s_2 (URI `ex:p2`) a score of 2, and triples in source s_3 (URI `ex:p4`) score 3.

Following Definition 14, a sorted access on a given join input allows to access partial bindings in descending score order. In a central data setting, a sorted access can be efficiently provided by using a score index over the input data [95]. In particular, [115] discusses implementation strategies for a sorted access for centrally stored RDF. Notice, while work on top-k join processing over distributed databases [51] aims at a similar setting, it also assumes such a complete index.

However, in the Linked Data context, only source statistics are assumed to be available, while the contained *triples are not indexed*, e.g., for the sake of result freshness. Following this tradition, we provide a lightweight sorted access, which only requires score bounds (computed at indexing time) for the sources, *while triples are ranked and sorted on-the-fly*.

Definition 18: Source Score Bounds

Given a source s with URI d , its *upper bound score* $\text{score}_u(d)$ is defined as the maximal score of the triples contained in \mathcal{T}_s^d :

$$\text{score}_u(d) := \max \{ \text{score}_Q(t) \mid t \in \mathcal{T}_s^d \} \quad (1a)$$

The *lower bound score* is defined as:

$$\text{score}_l(d) := \min \{ \text{score}_Q(t) \mid t \in \mathcal{T}_s^d \} \quad (1b)$$

For each triple pattern in the source index, we store its list of relevant sources in descending order of their upper bound scores score_u . This allows sources for each union operator to be retrieved sequentially in the order of their upper bound scores. As triples for a given source are not sorted, we replace each source-scan operator with a source-scan-sort operator. A source-scan-sort operator, after retrieving a source with URI d , sorts its triples \mathcal{T}_s^d according to their scores.

Example 21

Figure 19 and Figure 20 are two physical query plans for the query in Figure 18. In contrast to Figure 19, Figure 20 shows an access plan with source-scan-sort operators, which provide a sorted access to the bindings for pattern tp_5 and tp_6 , respectively.

However, if two (or more) sources, say, d_i and d_j , have overlapping source score bounds ($\text{score}_l(d_i) < \text{score}_u(d_j) < \text{score}_u(d_i)$), and both are inputs for

the same union, the output of the union will not be ordered if these sources are retrieved individually. We address this problem by treating both sources as “one source”. That is, sources with URIs d_i and d_j are scanned and sorted via the same source-scan-sort operator.

Note that $\text{score}_u(d)$ and $\text{score}_l(d)$ do not have to be precise – they could be approximated, e.g., based on expected scores.

Algorithmus 2 : LD-PBRJ.push(b)

Input : Pushed partial binding b on input $i \in \{i_1, i_2\}$.
Param. : Bounding strategy \mathcal{BS} .
Buffer : Output buffer (priority queue) \mathbf{O} . Buffer \mathbf{H}_i and \mathbf{H}_j for “seen” bindings from input i and j , respectively.

```

1 begin
2   if  $i = i_1$  then
3      $j \leftarrow i_2$ 
4   else
5      $j \leftarrow i_1$ 
6   Insert  $b$  into buffer  $\mathbf{H}_i$ 
7    $\mathbf{O}' \leftarrow$  Probe  $\mathbf{H}_j$  for valid join combinations with  $b$ 
8   foreach join result  $b' \in \mathbf{O}'$  do
9     Insert  $b'$  into  $\mathbf{O}$ 
10   $\beta \leftarrow \mathcal{BS}.\text{update}(b)$ 

```

3.3.2 Push-based Top-k Join Processing

Exploiting the ranking function, source index, and our sorted access, we can extend top-k strategies to the Linked Data setting.

Previous work on top-k join processing uses a pull-based query execution. That is, join operators actively pull on their inputs in order to produce an output [51, 93, 144]. We extend the PBRJ template in Algorithm 1 to allow for a *push*-based execution – well-suited for the Linked Data setting. For simplicity, the following presentation of the PBRJ algorithm uses *binary* joins. However, our algorithms can also be applied for general n -ary joins.

In a pull-based implementation, operators call a next method on their input operators to obtain new data. Consider also the generic iterator description in Section 2.2.4 (see Figure 12). In a push-based execution the control flow is inverted. That is, operators have a push method that is called by their input operators. Algorithm 2 shows the push method of the LD-PBRJ operator. The input from which the partial binding b was received is identified by $i \in \{i_1, i_2\}$.

- On Line 6 (Algorithm 2), the partial binding b is inserted into the buffer \mathbf{H}_i . Algorithm 2 features two buffers, \mathbf{H}_i and \mathbf{H}_j , for “seen” bindings, i.e., bindings which have been pushed by input i and j .
- On Line 7, we probe the other input’s buffer \mathbf{H}_j for valid (i.e., the join condition holds) join combinations. Successful join results are added to the

output buffer \mathbf{O} in Lines 8-9. Output buffer \mathbf{O} is a priority queue such that the result with the highest score is first.

- On Line 10, the threshold β is updated using the bounding strategy \mathcal{BS} . This provides an upper bound β on the scores of “unseen” (future) join results, i.e., join results comprising “unseen” partial bindings. When a join result in queue \mathbf{O} has a score \geq the threshold β , we know that there is no unseen (future) result, which has a higher score. Thus, this result in buffer \mathbf{O} is ready to be reported to a subsequent operator. If buffer \mathbf{O} contains k results, which are ready to be reported, the algorithm stops reading inputs (early termination, see Section 2.3.1).

As discussed in Algorithm 1, the standard PBRJ has two parameters: the *bounding strategy* \mathcal{BS} (see Definition 15, p. 36) and the *pulling strategy* \mathcal{PS} (see Definition 16, p. 36).

For the former parameter, the *corner-bound* strategy [93], is employed by many works and is also used in our approach.

The latter parameter, however, is intended for a pull-based query execution – thus, it can not be employed. Similar to the idea behind the pulling strategy, we aim to have control over the results that are pushed to subsequent operators. Because a push-based join has no influence over the data flow (within the query plan), we introduce a *scheduling strategy* to regain control. Furthermore, the push method in Algorithm 2, only adds join results to the output queue \mathbf{O} , but does not push them to a subsequent operator. Instead, the pushing is performed in a separate activate method (see Algorithm 4) – as mandated by the scheduling strategy.

Example 22

In Figure 20 the scheduling strategy uses the current query execution state to decide which operator to activate. In this example, the scheduling strategy activates the sorted access sa_5 , i.e., source-scan-sort, selection, and union for triple pattern tp_5 . This way, any binding for tp_5 , which is ready to be reported, will be pushed in the LD-PBRJ operator by the sorted access sa_5 .

We will provide further details on both strategies in the next paragraphs.

Bounding Strategy. A bounding strategy \mathcal{BS} is used to update the current score threshold β , i.e., the upper bound on scores of unseen join results. Since only those results in the output buffer \mathbf{O} can be reported that have a score equal to or greater than the threshold β , it is essential that the threshold β is as accurate (tight) as possible. In our experiments, we employed the well-known corner-bound strategy (see Definition 15, p. 36), as well as an improved variant of it, which we present in Section 3.3.3. However, it is important to note that any other strategy may also be applied for our LD-PBRJ algorithm.

Algorithmus 3 : LD-PBRJ.execute(\mathcal{Q}, k)

Input : Query \mathcal{Q} . Number of results k .
Param. : Scheduling strategy \mathcal{S} .
Data : Query plan P .
Output : Top- k query results in output buffer \mathbf{O} .

```

1 begin
2    $P \leftarrow \text{plan}(\mathcal{Q})$ 
3    $op \leftarrow \mathcal{S}.\text{nextOp}(P)$ 
4   while  $|\mathbf{O}| < k \wedge op \neq \text{null}$  do
5      $op.\text{activate}()$ 
6      $op \leftarrow \mathcal{S}.\text{nextOp}(P)$ 
7   return  $\mathbf{O}$ 

```

Scheduling Strategy. Deciding which input to pull from has a large effect on query processing performance [93, 144]. Previously, this decision was captured in a pulling strategy employed by the join operator implementation.

However, in push-based systems, the execution is not driven by operators, but by the input data. Join operators are only activated if input is actively pushed from operators lower in the operator tree. Therefore, instead of pulling, we propose a *scheduling strategy* that determines which operators in a query plan are scheduled for execution. That is, we move the control, over what input produces new partial bindings, from the join operator to the query engine, which orchestrates the query execution.

Algorithm 3 shows the execute method that takes a query \mathcal{Q} and the number of results k as input and returns the top- k results.

- We obtain a query plan P from the plan method on Line 2.
- We then use the scheduling strategy \mathcal{S} to determine the next operator that should be scheduled for execution (see Line 3).
- The scheduling strategy uses the current execution state as captured by the operators in the query plan to select the next operator. We activate the selected operator on Line 5.
- Last, a new operator is selected (see Line 6), until we either have obtained the desired number of k results or there is no more operator to be activated (i.e., all inputs have been retrieved and processed, see Line 4).

Algorithmus 4 : LD-PBRJ.activate()

Input : Subsequent operator op according to query plan P .
Buffer : Output buffer \mathbf{O} .
Data : Score threshold β .

```

1 begin
2   while  $\text{score}_{\mathcal{Q}}(\mathbf{O}.\text{peek}()) \geq \beta$  do
3      $b \leftarrow \mathbf{O}.\text{dequeue}()$ 
4      $op.\text{push}(b)$ 

```

Algorithm 4 depicts the activate method for our LD-PBRJ operator, which is called by the execute method. Intuitively, the activate method triggers a

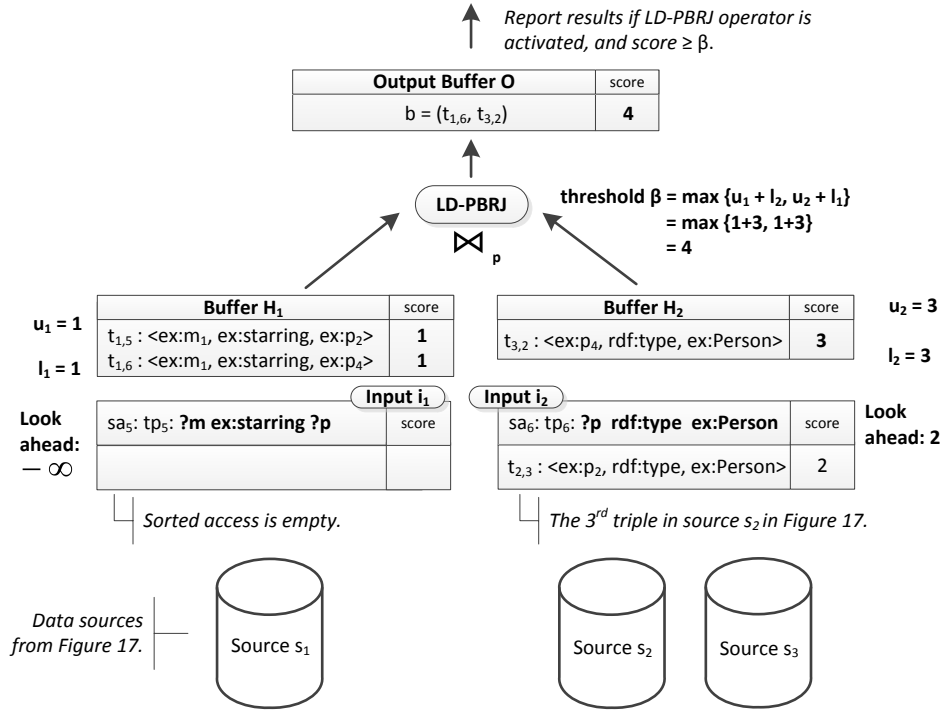


Figure 21: Detailed view on the LD-PBRJ operator from Figure 20. Data is retrieved from the sources in Figure 17.

“flush” of the operator’s output buffer **O**. That is, all computed results having a score larger than or equal to the operator’s threshold β (Line 2) are reported to the subsequent operator (Lines 3-4). An activate method for a source-scan-sort operator of a source d simply pushes all triples in d in a sorted fashion. Further, activate for selection and union operators causes them to push their outputs to a subsequent operator.

The question remains *how* a scheduling strategy should select the next operator (nextOp method, see Algorithm 3 at Line 6). For this, we apply the idea behind the state-of-the-art pulling strategy [144] (see Definition 16, p. 36) to perform *corner-bound-adaptive* scheduling. Basically, we choose the input which leads to the highest reduction in the corner-bound:

» Definition 19: Corner-Bound-Adaptive Scheduling

Given a LD-PBRJ operator, we prefer the input that could produce join results with the highest scores. That is, we prefer input i_1 iff $u_1 \oplus l_2 < u_2 \oplus l_1$ and i_2 otherwise. In case of ties, the input with the smaller number of unseen (future) partial bindings is chosen.

\oplus is the aggregation function used in the ranking function. Further, u_i is the highest score observed from input i , while l_i is the lowest observed score on input i .

The scheduling strategy then recursively selects and activates operators that may provide partial bindings for the preferred input. That is, in case the chosen input is another LD-PBRJ operator, which has an empty output queue, the

scheduling strategy selects and activates operators for its preferred input in the same manner.

Example 23

Figure 21 provides a detailed view on the LD-PBRJ operator from Example 22 in Figure 20. Further, we use data from sources s_1 , s_2 , and s_3 in Figure 17. For simplicity, we assume $k = 1$ and denote the j^{th} triple in source s_i as $t_{i,j}$. For instance, $t_{1,5}$ refers to the 5th triple in source s_1 (see Figure 17):

$$t_{1,5} = \langle \text{ex:m1}, \text{ex:starring}, \text{ex:p2} \rangle$$

Our scheduling strategy prefers input i_1 and selects/activates the source-scan-sort(s_1), $\sigma(\text{tp}_5)$, and union(tp_5). Note, also input i_2 would have been a valid choice, since the threshold β is not set yet. The LD-PBRJ reads $t_{1,5}$ and $t_{1,6}$ as new partial bindings from union(tp_5) and both bindings are inserted into \mathbf{H}_1 ($u_1 = l_1 = 1$). The scheduler now prefers input i_2 , because input i_1 is exhausted and selects/activates source-scan-sort(s_3), $\sigma(\text{tp}_6)$, and union(tp_6), because source s_3 has triples with higher scores than source s_2 . Then, union(tp_6) pushes $t_{3,2}$ and u_2 as well as l_2 is set to $\text{score}_Q(t_{3,2}) = 3$. Employing a summation as aggregation function \oplus , the threshold β is:

$$\beta = \max\{1 + 3, 1 + 3\} = 4$$

Then, $t_{3,2}$ is inserted into \mathbf{H}_2 . Joins between $t_{3,2}$ and bindings in \mathbf{H}_1 are attempted: $t_{1,6} \bowtie t_{3,2}$ yields a result b , which is then inserted into the output queue, \mathbf{O} . Finally, since $\text{score}_Q(b) = 4 \geq \beta = 4$ holds, b is reported as the top-1 result, and the LD-PBRJ terminates. Note, not all inputs have been processed. That is, source s_2 has not been scanned.

3.3.3 Improved Threshold Estimation

In the next paragraphs, we present two modifications of the corner-bound bounding strategy (see Definition 15, p. 36), which allow to calculate a *tighter* threshold $\tilde{\beta}$, thereby achieving earlier termination and result reporting, respectively.

Entity Query Bounds. A entity query comprises a set of triple patterns Q_v , which share a common variable at the subject position:

» Definition 20: Entity Query

An *entity query* Q_v is a directed labeled graph $\mathcal{G}_v^Q = (\mathcal{V}_v^Q, \mathcal{E}_v^Q)$, where \mathcal{V}^Q is the disjoint union $\mathcal{V}_v^Q = \mathcal{V}_V^Q \uplus \mathcal{V}_C^Q \uplus \mathcal{V}_K^Q$, with \mathcal{V}_V^Q as a set of variable nodes, \mathcal{V}_C^Q as a set of constants, and \mathcal{V}_K^Q as a set of user-defined keywords.

Further, $\mathcal{E}_v^Q = \{\langle v, p, o \rangle\}$, where v is a fixed entity variable from \mathcal{V}_V^Q , $p \in \ell_a \uplus \ell_r$, and $o \in \mathcal{V}_V^Q \uplus \mathcal{V}_C^Q \uplus \mathcal{V}_K^Q$.

A query Q can be conveyed as as disjoint union of its entity queries:

$$\biguplus_{Q_v \in Q_E} Q_v = Q$$

with Q_E as set of all entity queries contained in query Q .

Given Linked Data, we observed that every result for an entity query Q_v is contained in one single source. This is because a result is an entity and information related to that entity comes exclusively from the one source, which represents that particular entity. Exploiting this knowledge, a more precise corner-bound for entity queries can be calculated.

More precisely, we can derive that, in order to be relevant, sources for Q_v must satisfy all triple patterns in Q_v (because they must capture all information for the requested entity). Given relevant sources for Q_v are denoted as \mathbf{D} and the source upper bound is given by $\text{score}_u(d)$ for $d \in \mathbf{D}$, the maximal possible score for results matching Q_v , $\text{score}_{\max}(Q_v)$, can be derived based on the maximum source upper bound, $\max_{d \in \mathbf{D}} \text{score}_u(d)$. Formally:

$$\text{score}_{\max}(Q_v) := \text{score}_{\max}(tp_1) \oplus \dots \oplus \text{score}_{\max}(tp_n), \text{ with} \quad (2a)$$

$$\text{score}_{\max}(tp_i) := \max_{d \in \mathbf{D}} \text{score}_u(d), \text{ for } i = 1, \dots, n \quad (2b)$$

where entity query $Q_v = \{tp_i\}_{i=1, \dots, n}$ and \oplus is the aggregation, which is defined for the ranking function. Equation 2b holds because:


- Every triple that contributes to a result for entity query Q_v must be contained in a source $d \in \mathbf{D}$.
- Thus, every contributing triple must have a score $\leq \max_{d \in \mathbf{D}} \text{score}_u(d)$.

Look-Ahead Bounds. The corner-bound strategy uses the last seen scores, l_i , of partial bindings to calculate the current threshold, β . We observed that, when an partial binding b_i is received by an operator on input i , the next partial binding b_i^{next} (and its score $\text{score}_Q(b_i^{\text{next}})$) is often already available in the pushing operator. This is because:

- ① Source-scan-sort operators materialize their complete output before pushing to subsequent operators.
- ② LD-PBRJ operators maintain an output queue that often contains more than one result with scores greater than or equal to the current threshold β .
- ③ Given a source d_i has been pushed by a source-scan-sort operator, the source score upper bound of the next source to be loaded, d_{i+1} , is available.

By using the score of the next, instead of the last seen partial binding, we can provide a more accurate threshold β , because we can estimate the maximal score of unseen bindings from that particular input more accurately. Formally, we define the look-ahead bounds \tilde{l}_i for input i as:

$$\tilde{l}_i := \begin{cases} \text{score}_Q(b_i^{\text{next}}) & \text{if the binding } b_i^{\text{next}} \text{ is available} \\ l_i & \text{otherwise} \end{cases} \quad (3)$$

 **Example 24**


We have two look-ahead bounds in Figure 21. For input i_1 , we know that $\tilde{l}_1 = -\infty$, because there are no more relevant sources for pattern tp_5 . With regard to input i_2 , we have $\tilde{l}_2 = 2$, since the next relevant source s_2 has a source upper bound of 2.

Threshold Calculation. By applying entity query bounds as well as look-ahead bounds, we can refine the corner-bound β as:

$$\tilde{\beta} := \max \begin{cases} \min\{u_1 + \tilde{l}_2, \text{score}_{\max}(\mathcal{Q})\} \\ \min\{u_2 + \tilde{l}_1, \text{score}_{\max}(\mathcal{Q})\} \end{cases} \quad (4)$$

with

$$\text{score}_{\max}(\mathcal{Q}) := \bigoplus_{Q_v \in \mathcal{Q}_E} \text{score}_{\max}(Q_v)$$

 **Example 25**

Continuing Example 24, we can compute an improved corner-bound as follows:

$$\begin{aligned} \tilde{\beta} &= \max \begin{cases} \min\{u_1 + \tilde{l}_2, \text{score}_{\max}(tp_5) + \text{score}_{\max}(tp_5)\} \\ \min\{u_2 + \tilde{l}_1, \text{score}_{\max}(tp_5) + \text{score}_{\max}(tp_6)\} \end{cases} \\ &= \max \begin{cases} \min\{1 + 2, 1 + 3\} & = 3 \\ \min\{3 + -\infty, 1 + 3\} & = -\infty \end{cases} \\ &= 3 \end{aligned}$$

In contrast, the standard corner-bound is $\beta = \max\{u_1 + l_2, u_2 + l_1\} = \max\{3 + 1, 1 + 3\} = 4$.

The following theorem shows $\tilde{\beta}$ to be *correct*:

• Theorem 1: Improved Corner-Bound is Correct

Bound $\tilde{\beta}$ is correct, i.e., there is no unseen result b with $\text{score}_{\mathcal{Q}}(b) > \tilde{\beta}$.

Sketch of Proof

Given a query \mathcal{Q} , bound $\tilde{\beta}$ is correct iff: (i) $\bigoplus_{\mathcal{Q}_v \in \mathcal{Q}_E} \text{score}_{\max}(\mathcal{Q}_v)$ provides a valid score upper bound for \mathcal{Q} 's results and (ii) \tilde{l}_i is a valid score upper bound for unseen partial bindings from input i .

- (i) Considering the former constraint, assume there is a result b with $\text{score}_{\mathcal{Q}}(b) > \text{score}_{\max}(\mathcal{Q})$. Then, there must be at least one entity result, b_e , for an entity query $\mathcal{Q}_v \in \mathcal{Q}_E$ such that:

$$\text{score}_{\mathcal{Q}}(b_e) > \text{score}_{\max}(\mathcal{Q}_v)$$

In this case, b_e can not come from one single source, but must be distributed over multiple sources. This is because $\text{score}_{\max}(\mathcal{Q}_v)$ is composed of $\max_{d \in \mathcal{D}} \text{score}_u(d)$, which provides a valid score upper bound for all triple pattern bindings in b_e that come for the same source d . However, b_e being distributed over multiple sources contradicts our initial assumption, i.e., all results for \mathcal{Q}_v are located at one source. Therefore, constraint (i) holds for every $\mathcal{Q}_v \in \mathcal{Q}_E$.

- (ii) Regarding the latter constraint, \tilde{l}_i is a valid score upper bound over partial bindings from input i , if there is no unseen binding b in input i with $\text{score}_{\mathcal{Q}}(b) > \tilde{l}_i = \text{score}_{\mathcal{Q}}(b_i^{\text{next}})$. However,

$$\text{score}_{\mathcal{Q}}(b_i^{\text{next}}) < \text{score}_{\mathcal{Q}}(b)$$

can not hold, as we have a sorted access over input i . Thus, constraint (ii) holds for every partial binding b in input i .

Overall, as constraints (i) and (ii) hold at all times, $\tilde{\beta}$ is correct ■

Moreover, we can show $\tilde{\beta}$ to be *tighter* than the standard corner-bound:

•• Theorem 2: Improved Corner-Bound is Tighter

Bound $\tilde{\beta}$ is tighter than corner-bound β , i.e., $\tilde{\beta} \leq \beta$ holds at all times.

Sketch of Proof

We wish to show that $\tilde{\beta}$ is tighter than β , i.e., $\tilde{\beta} \leq \beta$ at all times. In order for $\tilde{\beta} \leq \beta$ to hold, either (i) $\text{score}_{\max}(\mathcal{Q}) \leq u_i \oplus l_j$, or (ii) $u_i \oplus \tilde{l}_j \leq u_i \oplus l_j$, must always hold.

- (i) Considering the former, since $\text{score}_{\max}(\mathcal{Q})$ merely provides a (valid) upper bound over scores of query results for a query \mathcal{Q} , there may be results for query \mathcal{Q} such that: $\text{score}_{\max}(\mathcal{Q}) > u_i \oplus l_j$ is true.

(ii) Thus, $\tilde{\beta} \leq \beta$ can only hold at all times, if $u_i \oplus \tilde{l}_j \leq u_i \oplus l_j$ always holds. Recall that \tilde{l}_j is always set to the “next” possible score, i.e., the score of the next partial binding seen in input j . As we have a sorted access, the “next” possible score is guaranteed to be smaller, thus, $\tilde{l}_j \leq l_j$ holds for every partial binding.

Overall, as (ii) holds at all times, $\tilde{\beta}$ is tighter than β ■

3.3.4 Early Pruning of Partial Results

Source information can be exploited to *prune partial results*, which can not contribute to final top-k results. This way, we aim at reducing space consumption of buffer \mathbf{O} and buffer \mathbf{H} in Algorithm 2. Smaller buffers, in turn, reduce join costs (due to less join attempts) and buffer maintenance costs.

The idea of pruning partial bindings has also been pursued by approximate top-k selection approaches [157]. However, in contrast to previous works, we do not approximate, but only prune those partial bindings that are *guaranteed not to contribute to the final top-k results*. Thus, we still compute exact top-k results – as shown in Theorem 3.

Intuitively, we prune a partial result, if its score together with the maximal possible score for its unevaluated query fragment, is smaller than the minimal (so far computed) top-k result score. Note, the opportunity for pruning only arises if at least k complete results have been computed.

More formally, let b be a partial binding for query \mathcal{Q} , with $\mathcal{Q}(b) \subseteq \mathcal{Q}$ as evaluated query fragment and $\mathcal{Q}^u(b) = \mathcal{Q} \setminus \mathcal{Q}(b)$ as unevaluated query fragment (see Definition 7, p. 22). Then, a score upper bound of all final results comprising b can be obtained by aggregating $\text{score}_{\mathcal{Q}}(b)$ and the maximal possible score of results for $\mathcal{Q}^u(b)$, $\text{score}_{\max}(\mathcal{Q}^u(b))$:

$$\text{score}_{\mathcal{Q}}(b') \leq \text{score}_{\mathcal{Q}}(b) \oplus \text{score}_{\max}(\mathcal{Q}^u(b)) \quad (5a)$$

with b' as a complete binding comprising partial binding b . Similar to Equation 4, $\text{score}_{\max}(\mathcal{Q}^u(b))$ can be computed as the aggregation of maximal source upper bounds, which are obtained for triple patterns in $\mathcal{Q}^u(b) = \{tp_1, \dots, tp_m\}$:

$$\text{score}_{\max}(\mathcal{Q}^u(b)) := \text{score}_{\max}(tp_1) \oplus \dots \oplus \text{score}_{\max}(tp_m), \text{ with} \quad (5b)$$

$$\text{score}_{\max}(tp_i) := \max_{d \in \mathbf{D}_i} \text{score}_u(d), \text{ for } i = 1, \dots, m \quad (5c)$$

where \mathbf{D}_i is the set of relevant sources for pattern $tp_i \in \mathcal{Q}^u(b)$ (according to the source index), and \oplus is the aggregation, which is defined for the ranking function.

Last, the following theorem can be established:

•→ **Theorem 3**

Given a query \mathcal{Q} , a partial binding b cannot contribute to one or more final top-k results for query \mathcal{Q} , if

$$\text{score}_{\mathcal{Q}}(b) \oplus \text{score}_{\max}(\mathcal{Q}^u(b)) \leq \min_{\bar{b} \in \mathbf{O}} \text{score}_{\mathcal{Q}}(\bar{b})$$

where \mathbf{O} is the output buffer in Algorithm 2, which contains at least k results.

Sketch of Proof

If a partial binding b is pruned, while actually contributing to a final top-k result b' , it must hold:

$$\text{score}_{\mathcal{Q}}(b') > \min_{\bar{b} \in \mathbf{O}} \text{score}_{\mathcal{Q}}(\bar{b})$$

That is, the score of b' is larger than the minimal score among the currently known complete results in \mathbf{O} . However, if b was pruned, it holds that:

$$\text{score}_{\mathcal{Q}}(b) \oplus \text{score}_{\max}(\mathcal{Q}^u(b)) \leq \min_{\bar{b} \in \mathbf{O}} \text{score}_{\mathcal{Q}}(\bar{b})$$

At the same time, as given in Equation 5a and Equation 5b, $\text{score}_{\max}(\mathcal{Q}^u(b))$ provides a valid upper bound of scores for b' :

$$\text{score}_{\mathcal{Q}}(b') \leq \text{score}_{\mathcal{Q}}(b) \oplus \text{score}_{\max}(\mathcal{Q}^u(b))$$

Therefore,

$$\text{score}_{\mathcal{Q}}(b') \leq \min_{\bar{b} \in \mathbf{O}} \text{score}_{\mathcal{Q}}(\bar{b})$$

which contradicts the initial assumption that b' is a top-k result ■

3.4 EVALUATION

In the following, we present our evaluation results and empirically validate:

- *Hypothesis 1 in Section 3.2*
Top-k processing via our LD-PBRJ operator outperforms state-of-the-art Linked Data query processing.
- *Hypothesis 2 in Section 3.2*
Our tighter bounding in Equation 4, as well as early pruning strategy in Equation 5a and Equation 5b, leads to a better performance than state-of-the-art top-k processing.

3.4.1 Evaluation Setting

Systems. We implemented three different systems, which are all based on push-based join processing. For every query, we generated a left-deep query plan with random join order. All systems use the same plan and differ solely in the join operator implementation.

First, we have the push-based symmetric hash join operator (*shj*) [105, 106], which does not employ top-k processing techniques, but instead produces all results, and then sorts them to obtain the desired top-k results. Further, we use two implementations of the LD-PBRJ operator. Both use the corner-bound-adaptive scheduling strategy, but employ different bounding strategies. The first uses the corner-bound (*rj-cc*) from previous work [144] (see Definition 15, p. 36), while the second (*rj-tc*) employs our optimization with tighter bounds and early result pruning, see Equation 4, Equation 5a, and Equation 5b.

*The *shj* baseline is used to study the benefits of top-k processing in the Linked Data setting, see Hypothesis 1. With regard to Hypothesis 2, we employ *rj-cc* to analyze the effect of the proposed optimizations.*

All systems were implemented in Java 6. Experiments were run on a Linux server with two Intel Xeon 2.80GHz Dual-Core CPUs, 8GB RAM and a Seagate ST31000340AS 1TB hard disk. Before each query execution, all operating system caches were cleared. The presented values are averages collected over three runs.

Dataset and Queries. We used queries from the Linked Data FedBench benchmark.²⁸ Due to schema changes in DBpedia²⁹ and time-outs observed during the experiments (> 2 min), three of the 11 FedBench queries were omitted. Additionally, we used 12 queries that we created. In total, we had 20 queries that differ in the number of their results (1 – 10K) and in their complexity in terms of the number of triple patterns (2 – 5). A complete listing of our queries can be found in the appendix, see Section A.1.

To obtain the dataset, we executed all queries directly over the Web of Linked Data using a link-traversal approach [78] and recorded all Linked Data sources that were retrieved during query execution. In total, we downloaded 681,408 Linked Data sources, comprising a total of 1,867,485 triples. From this dataset we created a source index that is used by the query planner to obtain relevant sources for the given triple patterns.

We observed that network latency greatly varies between hosts and evaluation runs. In order to systematically study the effects of top-k processing, we decided to store the sources locally and to simulate Linked Data query processing on a single machine – as done before [105, 106].

Parameters. Parameter $k \in \{1, 5, 10, 20\}$ denotes the number of top-k results to be computed. Further, we employed three different score distributions $d \in \{u, n, e\}$ (uniform, normal and exponential).

More precisely, scores were randomly assigned to triples in the dataset. We applied three different score distributions: uniform, normal ($\mu = 5, \sigma^2 = 1$) and

²⁸<http://fedbench.fluidops.net>, retrieved 2013-12-07.

²⁹<http://dbpedia.org>, retrieved 2013-12-07.

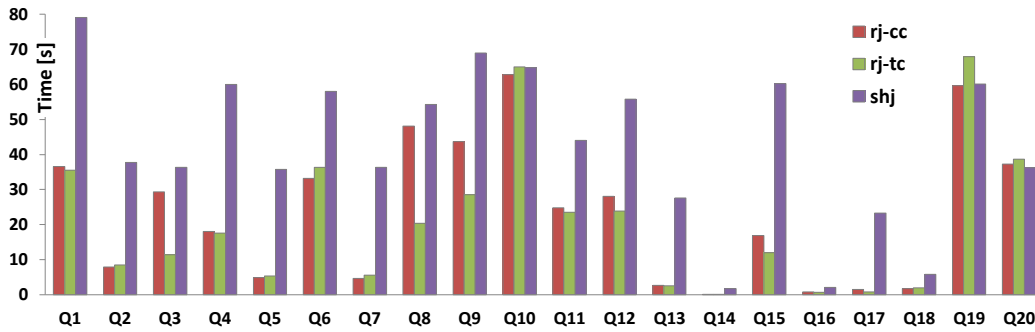


Figure 22: All queries with their evaluation times ($k = 1$, $d = n$).

exponential ($\lambda = 1$). This allows us to abstract from a particular ranking function and examine the applicability of top- k processing for different classes of ranking functions. We used a summation as score aggregation function, \oplus .

3.4.2 Evaluation Results

Overall Results. Figure 22 shows an overview of processing times for all queries ($k = 1$, $d = n$). We can see that the LD-PBRJ approaches (rj-tc and rj-cc) perform better or at least equal to shj for all queries. Note, we discuss outlier queries Q19 and Q20 in the following. On average, the execution times for rj-cc and rj-tc are 23.13s and 20.32s, whereas shj required 43.05s for query execution. This represents a performance improvement of the rj-cc and rj-tc operators over the shj operator by factors of 1.86 and 2.14, respectively.

The improved performance of the LD-PBRJ operators is because of their top- k processing. That is, LD-PBRJ operators do not have to process all input data in order to produce the k top results, but can terminate early. In contrast, the shj operator produces all results. Figure 23-a shows the average number of retrieved sources for different values of k . We can see that the LD-PBRJ approaches retrieve fewer sources than the baseline approach. In fact, rj-cc and rj-tc retrieve and process only 41% and 34% of the sources that the shj approach requires. This is a significant advantage in the Linked Data context, where sources can only be retrieved in their entirety.

However, we also see that the LD-PBRJ operators sometimes do not perform better than the shj operator. In these cases, the result is small, e.g., Q19 has only two results. The LD-PBRJ operators have to read all inputs and compute all results in these cases. For example, for Q20 the LD-PBRJ approaches retrieve and process all 35,103 sources – just as the shj approach does.

Bounding Strategies. Next, let us examine the effect of the bounding strategies on overall execution time. rj-cc and rj-tc require an average processing time of 23.13s and 20.32s, respectively. This is an improvement of 12% of rj-tc over rj-cc. For some queries, e.g., Q3, the improvement is even higher. Given query Q3, rj-tc requires 11s – compared to 30s for rj-cc.

The improved performance can be explained with the tighter, more precise bounding strategy realized by rj-tc. For instance, given query Q3, our bounding

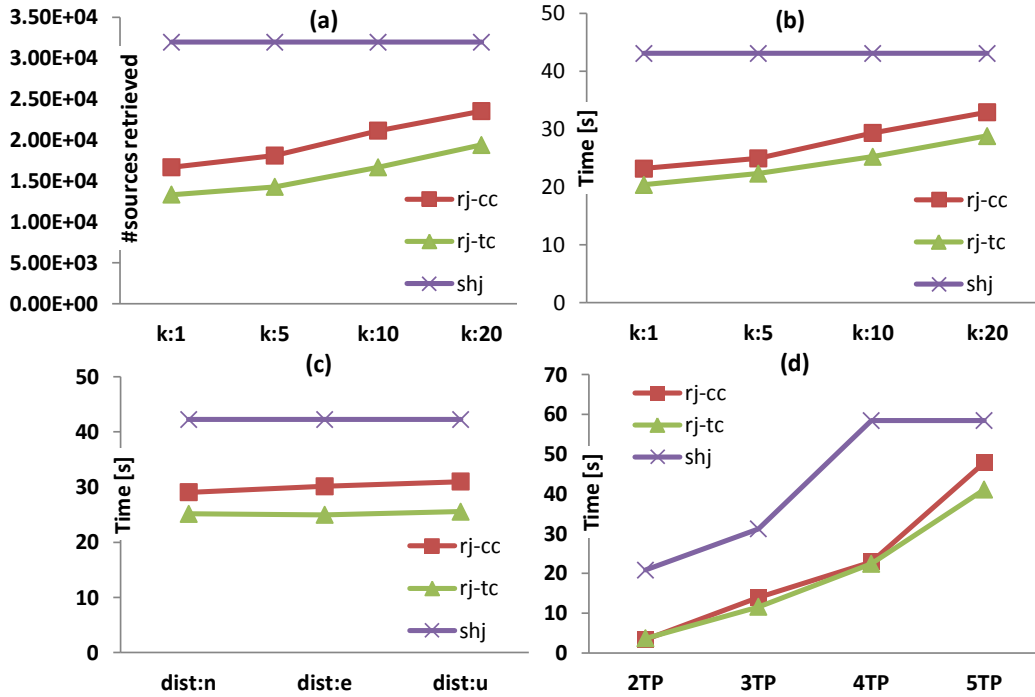


Figure 23: (a) Average number of sources over all queries vs. different k ($d = n$). (b) Average evaluation time over all queries vs. different k ($d = n$). (c) Average evaluation time over all queries vs. different score distributions ($k = 10$). (d) Average evaluation time over all queries vs. varying number of triple patterns ($k = 1$, $d = n$).

strategy can take advantage of a large star-shaped subexpression with 3 patterns in Q3 – leading to an accurate entity query bound. Additionally, we observed that the look-ahead strategy helps to calculate a much tighter upper bound, especially when there are large score differences between successive bindings from a particular input.

A tighter (more precise) bound means that results can be reported earlier and less inputs have to be read. This is directly reflected in the number of sources that are processed by rj-tc and rj-cc. On average rj-tc requires 23% fewer sources than rj-cc. Note, while in Figure 22 rj-tc’s performance often seems to be comparable to rj-cc, Figure 23-a makes the differences more clear in terms of the number of retrieved sources. For example, both systems require an equal amount of processing times for Q17. However, rj-tc retrieves 7% less sources. Such “small” savings did not show properly in our evaluation (since we retrieved sources locally from disk), but would effect processing time in a real-world setting with network latency.

Concerning the outlier Q19, we noticed that rj-tc did read slightly more input (2%) than rj-cc. This behavior is due to our implementation: Sources are retrieved in parallel to join execution. In some cases, the join operators and the source retriever did not stop at the same time.

We conclude that rj-tc performs equally well or better than rj-cc. For queries with large entity query fragments or with inputs, which have large score differ-

ences between successive sources, we are able to achieve performance gains of up to 60% by means of the `rj - tc` operator.

Early Pruning. We observed that the pruning strategy leads to lower buffer sizes, i.e., less memory consumption. For instance, given query Q9, the `rj - tc` operator could prune 8% of its buffered data. However, we also noticed that the number of sources loaded/scanned is actually the key performance factor. While pruning had positive effects, the improvement is small compared to what could be achieved with tighter bounds. For example, for query Q9 73% of total processing time was spent on loading and scanning sources.

Effect of Result Size k . Figure 23-b depicts the average query processing time for all three approaches at different k (with score distribution $d = n$). We observed that the time for `shj` is constant in k , since `shj` always computes all results. Further, we observed that the LD-PBRJ, `rj - tc` and `rj - cc`, approaches outperform `shj` for all k . However, with increasing k , more inputs need to be processed. Thus, the runtime differences between the LD-PBRJ approaches and `shj` operator become smaller. For instance, for $k = 1$ the average time saving with regard to `shj` is 46% (52%) for `rj - cc` (`rj - tc`). Given $k = 10$, the average time saving with regard to `shj` is only 31% (41%) for `rj - cc` (`rj - tc`).

Further, we can see in Figure 23-b that `rj - tc` outperforms `rj - cc` over all values for k . The differences are due to our tighter bounding strategy, which substantially reduces the amount of required inputs. For example, for $k = 10$, `rj - tc` requires 21% less inputs than `rj - cc`.

Last, we noted that `rj - tc` and `rj - cc` behave similarly for increasing k . That is, both operators become less efficient with increasing k , see Figure 23-b.

Effect of Score Distributions. Figure 23-c shows average processing times for all approaches for the three score distributions. We see that the performance of both LD-PBRJ operators varied only slightly with regard to different score distributions. For instance, `rj - cc` performed 7% better on the normal distribution compared to the uniform distribution. The `shj` operator has constant evaluation times over all distributions. We argue that this shows the general applicability of our LD-PBRJ approach. That is, its performance gains are not dependent on a particular ranking function.

Effect of Query Complexity. Figure 23-d shows average processing times (with $k = 1, d = n$) for different numbers of triple patterns. Overall, processing times increase for all systems with an increasing number of patterns. Again, we see that the LD-PBRJ operators, `rj - tc` and `rj - cc`, outperform `shj` for all query sizes. In particular, for 5 query patterns, we noticed the effects of our entity bounds more clearly, as those queries often contained entity queries up to length 3.

3.5 RELATED WORK

3.5.1 *Pull-based, Centralized Top-k Processing*

The top-k join problem has been addressed by many works – as discussed in the comprehensive survey [95]. Most notably, the J^* rank join, based on the A^* algorithm, was proposed in [125]. Other top-k join algorithms, the HRJN and the HRJN*, were introduced in [93] and further extended in [109].

In contrast to such works, we aim at a Linked Data context. As recent works [78, 75, 105, 106] have shown, Linked Data query processing introduces various novel challenges. In particular, instead of a *pull*-based top-k join, we needed a push-based execution for queries over Linked Data [105, 106]. We therefore extended the well-known PBRJ framework to allow for a push-based query execution.

The majority of top-k join solutions target centrally stored data [95]. A prominent example is the Pull/Bound Rank Join framework [144], which captures the existing top-k join approaches within one single framework. However, as we outlined in Section 1.2.3, Web data is inherently distributed over a vast space of sources. Centrally indexing these sources would come at great costs in terms of index maintenance and storage space. Moreover, central indexing would require allowed data access to all Web data sources. This strongly restricts the possible use-cases of Web data, e.g., in a commercial application.

Note that the only other work, which extends top-k join processing to RDF data is [115]. We presented a top-k join processing approach for *distributed* RDF. That is, data graphs are spread over a space of sources. In contrast, data may be stored and indexed centrally. For this, [115] proposed an extension of the SPARQL algebra, the so-called SPARQL-RANK, as well as an optimized SPARQL-RANK implementation for RDF stores. This work is *complementary to our solution*, since [115] focuses on a pull-based top-k join operator, which is well-suited for a centralized RDF store. However, the authors do not consider the case of data being located at various (small) data sources.

Last, different bounding strategies have been proposed. In [55, 144], the authors introduced a new Feasible-Region (FR) bound for the general setting of n -ary joins and multiple score attributes. However, it has been shown that the PBRJ template with corner-bound is *instance-optimal* in the restricted setting of binary joins and a single score attribute [55, 144]. We extend the corner-bound to the Linked Data setting and provide tighter/more precise bounds, which allow for earlier termination and better performance.

3.5.2 *Distributed Top-k Processing*

There has been work on distributed selection top-k processing, e.g., [20, 120, 126, 163, 170], and distributed aggregation top-k processing, e.g., [131, 138]. Unfortunately, the selection top-k problem and the aggregation top-k problem are both highly different from the join top-k problem. The selection top-k problem aims at finding top-ranked entities, where each entity is ranked according to multiple dimensions. The aggregation top-k problems aims at finding top-ranked sets of

bindings, where each set is ranked according to an aggregation function. Thus, extending such techniques to a join top-k problem is not straightforward [51].

With regard to the top-k join problem, [51] presents a top-k join approach for distributed databases, while [6, 175, 176] target a P2P scenario. We differ from the former, as we rely exclusively on simple HTTP lookups for data access as well as use only basic indexes in the form of the source index [51]. Moreover, for each Linked Data source we only require a min/max score bound, whereas [51] utilizes complete histograms over scoring attributes. The latter works rely on score sampling to compute a lower score bound [6, 175, 176]. However, given many small sources, such a solution would lead to prohibitive costs. Overall, we aimed at a lightweight sorted access implementation, which requires only simple score statistics. In turn, we have only minimal maintenance – a key advantage with regard to the highly dynamic Web of data.

3.5.3 Approximate Top-k Processing

Various works target the computation of approximate top-k results [15, 16, 120, 151, 157]. In particular, similar to our pruning approach, [157] estimates the likelihood of partial bindings contributing to a final result. If this estimate is below a given threshold, partial bindings are pruned. However, [157] addressed the selection top-k problem, which is different to our top-k join problem. More importantly, we do not rely on probabilistic estimates for pruning, but employ accurate upper bounds. Thus, we do not approximate the final top-k results.

3.6 SUMMARY

In this chapter, we addressed the first research question:

🔗 Research Question 1

How to enable top-k query processing on highly distributed, schemaless Web data?

For this, we validated Hypothesis 1 and Hypothesis 2:

✅ Hypothesis 1

Join top-k processing based on the Pull Bound Rank Join (PBRJ) framework (see Algorithm 1) can be extended to match the requirements of highly distributed Web data. Moreover, such a top-k processing allows for significant performance gains for computation of ranked results over Web data.

Targeting above hypothesis, we proposed a novel LD-PBRJ framework. We implemented our system and empirically showed its feasibility and performance advantages.

✓ Hypothesis 2

Given our lightweight sorted accesses, we can improve the state-of-the-art bounding strategy (corner bound, see Definition 15, p. 36). Moreover, the LD-PBRJ framework can be extended to allow for pruning of partial bindings, which cannot lead to a complete top-k binding. This way, we process less partial bindings and save computation time due to less join attempts.

With regard to Hypothesis 2, we provided a theoretical analysis – thereby validating our proposed bounding strategy. Moreover, we validated this hypothesis by means of our experiments. In fact, we could show that our improved bounding strategy can lead to significant performance gains.

In the next chapter, we will present an approach for selectivity estimation of hybrid queries over Web data. By means of this work, we allow query optimizers to construct suitable query plans, which comprise (amongst other operators) our LD-PBRJ operator.

SELECTIVITY ESTIMATION

SELECTIVITY ESTIMATION

Context of this Chapter. In this chapter, we present a *selectivity estimation for text-rich RDF data*, which is based on our previous publication [3]. For this, we introduce an approach, which compactly summarizes structured/unstructured data and exploits this summary to estimate the result size of a hybrid query.

We aim to *effectively* and *efficiently* estimate result set sizes for hybrid queries over Web data. The former goal refers to accurate estimations, while the latter goal refers to efficiency in terms of time and space needed for selectivity estimations. Our work contributes to this thesis with regard to two aspects:

- ❶ *We employ selectivity estimation as means to compute the binding probability in our approximate top-k join approach, see Section 5.3.* More specifically, we exploit selectivity estimation to calculate the probability for a partial binding to contribute to one or more complete bindings. We use this probability to prune such partial bindings that have a low chance of contributing to a complete binding – thereby approximating the top-k result set. Note, this approximate query processing targets our Research Question 4: How to enable approximate rank-aware query processing on Web data?
- ❷ Selectivity estimation is a crucial part of query optimization. Figure 24 illustrates the traditional query processing pipeline – we highlighted the steps, where selectivity estimation plays a key role. More specifically, by means of selectivity estimation, costs for physical query plans can be approximated and a query optimizer can construct an optimal physical plan. In particular, selectivity estimation allows to compare query plans featuring different join orders with each other. The join order is critical for hybrid queries, since some patterns may be highly selective, while others may bind large amounts of triples. Thus, it is crucial for an optimizer to construct a physical query plan, which minimizes the number of intermediate query results.

Moreover, works on rank-aware query processing showed the need for *depth estimation* for PBRJ operators [94, 145, 146]. Depth estimation is concerned with approximating the number of partial bindings read from a particular PBRJ operator input. As discussed in [94, 145, 146], such information is crucial for integration of PBRJ operators in physical query plans. For this, recent work in [145, 146] presented the DEEP framework, which offers a flexible algorithm template for depth estimation, based on selectivity estimations techniques.

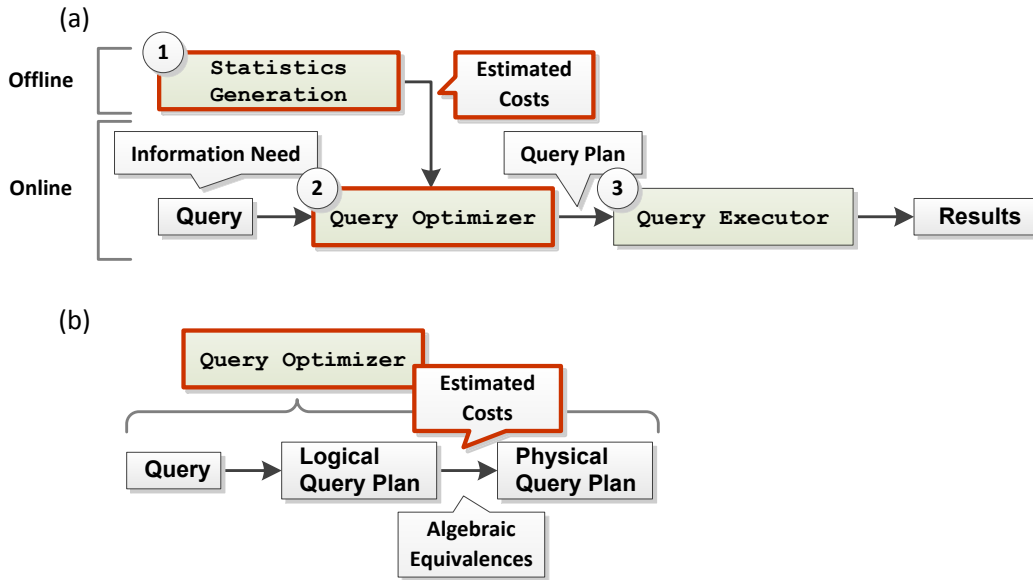


Figure 24: Context of our selectivity estimation for text-rich RDF data approach [149]. Figure (b) provides a detailed view on the query optimizer, which is also captured in figure (a). Steps in which our selectivity estimation plays an important role are highlighted.

Our selectivity estimation approach can be directly applied for both optimizations. This way, query optimization is tailored towards the characteristics of hybrid queries and Web data – leading to better physical query plans/performance gains. In particular, employing the DEEP framework in conjunction with our selection estimation would allow an optimal integration of the LD-PBRJ operator as well as the A-PRBJ operator in physical query plans. Note, the A-PRBJ operator is an approximate top-k join operator that is introduced in Section 5.3.

Outline. In Section 4.1, we motivate the problem of selectivity estimation over text-rich RDF data, and introduce foundations as well as our terminology. We discuss research problems and our contributions in Section 4.2. We present our main approach, the BN^+ system, in Section 4.3. Then, we discuss evaluation results in Section 4.4. In Section 4.5, we give an overview of related works. Last, we conclude with Section 4.6.

4.1 INTRODUCTION

4.1.1 Motivation

The use-cases for selectivity estimation techniques within a query processing context are manifold:

- ① Processing hybrid queries requires a *query optimizer*, which relies on *selectivity estimates* to approximate query execution costs, which are associated with a given physical query plan.

More specifically, the cost model commonly captures costs for computation (e.g., the number of join attempts), costs for storage access (e.g., the number of triples loaded), or costs for memory (e.g., the number of buffered partial bindings in join operators). These costs can be estimated via selectivity estimation approaches. See Section 2.2.3 for a detailed introduction to the cost model.

By means of this cost estimation, the query optimizer searches, e.g., using a dynamic programming technique [53], over a space of possible physical query plans – targeting an *optimal physical query plan*. Such an optimal query plan optimizes the overall query execution costs. In particular, the optimal plan aims at minimizing the amount of intermediate results – the driving factor behind the overall query execution time [53]

- ② As a special case of cost estimation, depth estimation has been proposed for rank-aware join operators [94, 145, 146]. Depth estimation is concerned with the estimation of the number of inputs, which are read from a join input, in order to produce the desired top-k results [94, 145, 146]. For this, previous works employed selection estimation techniques.

Similar to above presented query execution costs, depth estimation is used by a query optimizer to construct an optimal physical query plan, which comprises rank-aware join operators [94, 145, 146].

- ③ Selectivity estimates, as studied in this chapter, are not only crucial for query optimization, but for many problems that can be solved via conjunctive queries. For instance, data extraction [164] and data integration programs [37] have been formulated as queries, which involve selection and (similarity) join patterns.

Selectivity estimations are mostly performed at *runtime*. Thus, efficiency of the required computations is essential. Targeting a low computational overhead, selectivity estimation is based on *data synopses*, which approximately capture underlying data value distributions through a statistical summary. Several assumptions are commonly employed to keep such a synopsis small and simple:

- ① *Uniform Distribution Assumption*

Let us come back to the example in Figure 7. Let X_{name} be a random variable for predicate name, with Ω as its *sample space*. The uniform distribution assumption implies that all values for a predicate are equally likely. In Figure 7 predicate name features five distinct values: $|\Omega(X_{name})| = 5$. Thus, the probability for an entity x having name “Gregory Peck” is:

$$P(X_{name} = \text{“Gregory Peck”}) \approx \frac{1}{|\Omega(X_{name})|} = \frac{1}{5}$$

In other words, the probability for a query pattern $tp = \langle x, name, \text{“Gregory Peck”} \rangle$ is $\frac{1}{5}$. Clearly, this assumption may lead to misestimates, when “Gregory Peck” is a common name shared by several entities.

② *Predicate Value Independence Assumption*

Similarly to X_{name} , let X_{comment} be a random variable for predicate comment in Figure 7. Given a second query pattern $tp = \langle x, \text{comment}, \text{"Audrey Hepburn was..."} \rangle$ with:

$$P(X_{\text{comment}} = \text{"Audrey Hepburn was..."}) = 1$$

the predicate value independence assumption dictates that the two predicate values are independent. That is, the probability of observing both events is:

$$\begin{aligned} P(X_{\text{name}} = \text{"Gregory Peck"}, X_{\text{comment}} = \text{"Audrey Hepburn was..."}) &\approx \\ P(X_{\text{name}} = \text{"Gregory Peck"}) \cdot P(X_{\text{comment}} = \text{"Audrey Hepburn was..."}) & \\ = \frac{1}{5} \cdot 1 & \end{aligned}$$

However, as we can observe in the data (see Figure 7), there is actually no entity that is associated with that name and comment. Thus, the joint probability should actually be 0. Such a misestimate is due to correlations among data values. Given the value for name, a particular value for comment is more or less likely to occur (instead of being equally likely).

③ *Join Predicate Independence Assumption*

Last, there is the join predicate independence assumption, which is a special case of the previous assumption. This assumption states that the existence of a predicate is independent of the value/existence of another predicate. Coming back to our example in Figure 7, the existence of comment and any value for name would be assumed to be independent. Again, such a simplification would lead to misestimates, since predicate comment only occurs with name "Audrey Kathleen Hepburn".

As demonstrated by our examples, above independence assumptions greatly effect the effectiveness (i.e., accuracy) of selectivity estimates. Inaccurate estimates, however, can greatly affect applications, which rely on those selectivity estimates. In particular, query optimizers will construct non-optimal physical query plans, which may lead to an expensive query execution.

A large body of work has been devoted to avoid one or more of the above independence assumptions. Approaches aim to consider data correlations, thereby achieving more accurate selectivity estimates. Assumption 1 is addressed by counting values and embedding the resulting frequency statistics into synopses such as histograms [135] and wavelets [118]. Dealing with Assumption 2 and Assumption 3 requires a joint distribution of two or more random variables, which may be approximated via join synopses [10], tuple-graph synopses [154] or probabilistic relational models (PRMs) [60, 162].

4.1.2 Selectivity Estimation

Given a query Q , its selectivity, denoted by $\text{sel}(Q)$, is defined as the cardinality of Q 's result set (see Definition 12, p. 29). In this work, we address the problem of estimating $\text{sel}(Q)$, which may be decomposed into two functions:

- The Function $\mathcal{R} : \mathcal{Q} \rightarrow \mathbb{N}$ gives an upper bound cardinality of the result set for query \mathcal{Q} .

In previous works [60], $\mathcal{R}(\mathcal{Q})$ is estimated as size of the cross-product of the tables, which the query \mathcal{Q} is evaluated over. In our setting, table names are not explicitly given in a query. To obtain $\mathcal{R}(\mathcal{Q})$, we consider an upper bound of results for every distinct query variable. That is, for each $v \in \mathcal{V}_{\mathcal{V}}^{\mathcal{Q}}$, we upper-bound the number of its bindings, $\mathcal{R}(v)$, as number of all entities belonging to class c :

$$\mathcal{R}(v) = |\{s \mid \langle s, \text{type}, c \rangle \in \mathcal{E}\}| \quad (6)$$

with \mathcal{E} as edge set of the data graph \mathcal{G} . However, computing $\mathcal{R}(v)$ like this requires \mathcal{Q} to contain a class pattern $\langle v, \text{type}, c \rangle$. If v has no class assigned, we use the number of all entities, $|\mathcal{V}_{\mathcal{E}}|$, as an estimate for $\mathcal{R}(v)$. Then, $\mathcal{R}(\mathcal{Q})$ is given by:

$$\mathcal{R}(\mathcal{Q}) := \prod_{v \in \mathcal{V}_{\mathcal{V}}^{\mathcal{Q}}} \mathcal{R}(v) \quad (7)$$

- The *probabilistic component* \mathcal{P} defines a probability function that maps \mathcal{Q} to a probability. More precisely, \mathcal{P} assigns a probability to a binary random variable, say $\mathbb{1}_{\mathcal{Q}}$, modeling whether or not \mathcal{Q} 's result set is non-empty. In other words, $\mathbb{1}_{\mathcal{Q}}$ captures whether \mathcal{Q} holds. We write $\mathcal{P}(\mathbb{1}_{\mathcal{Q}} = \mathbf{T})$ as $\mathcal{P}(\mathcal{Q})$ for simplicity.

The probabilistic component $\mathcal{P}(\mathcal{Q})$ captures the joint probability over a set of random variables – one random variable for each query pattern in \mathcal{Q} . Intuitively, each such random variable models whether the associated query pattern holds or not.

Employing function \mathcal{R} and the probabilistic component $\mathcal{P}(\mathcal{Q})$, we can refine the informal definition in Definition 12:

» **Definition 21: Selectivity Estimation Function**

Given a query \mathcal{Q} and a data graph \mathcal{G} , a selection estimation function, sel , is defined as:

$$\text{sel}(\mathcal{Q}) := \mathcal{R}(\mathcal{Q}) \cdot \mathcal{P}(\mathcal{Q}) \quad (8)$$

In the next paragraphs, we introduce Bayesian networks (BNs) and their template-based representation, in order to efficiently and effectively compute $\mathcal{P}(\mathcal{Q})$ for a given query \mathcal{Q} .

4.1.3 Probabilistic Framework – Bayesian Networks

In order to compute the query probability $\mathcal{P}(\mathcal{Q})$ for a query \mathcal{Q} with $|\mathcal{Q}| = n$, we need to estimate a joint probability over n random variables: X_1, \dots, X_n . Assuming each random variable X_i has a sample space Ω_i , capturing the exact

joint probability would require $\prod_i |\Omega_i|$ space. To reduce space complexity, independence assumptions are commonly employed. One might assume all random variables to be pairwise *independent*, thereby reducing the space to $\sum_i |\Omega_i|$. However, as discussed above, such independence assumptions rarely result in accurate joint probability approximations. Instead, *conditional independence* has been exploited to obtain a more effective, yet scalable, approach for probability estimation [48, 60, 162]. This is based on the observation that often correlations between two random variables X_1 and X_2 can be “mediated” by a third variable X_3 . That is, variables X_1, X_2 become independent given X_3 :

$$X_1 \perp X_2 \mid X_3$$

Finding and exploiting such conditional independences enables a factorization of the full joint distribution – hence, allows a more compact representation and efficient computation of probabilities.

A Bayesian network (BN) represents a directed *graphical model* that allows for a compact representation of joint distributions by means of two components: a *network structure* and model *parameters* [102].

- *BN Network Structure*

The BN network structure is a directed acyclic graph, where nodes stand for random variables and edges represent dependencies among them. Given parent nodes $\text{Pa}(X_i) = \{X_j, \dots, X_k\}$, a random variable X_i is dependent on $\text{Pa}(X_i)$, but conditionally independent of all non-descendant nodes (random variables), i.e., all nodes which are not reachable from X_i when removing $\text{Pa}(X_i)$.

- *BN Parameters*

BN parameters comprise conditional probability distributions (CPDs) for random variables in the network. That is, each node X_i is associated with a CPD capturing the conditional probability $P(X_i \mid \text{Pa}(X_i))$.

A BN allows for computing the joint distribution $P(X_1, \dots, X_n)$ via the chain rule [102]:

$$P(X_1, \dots, X_n) \approx \prod_i P(X_i \mid \text{Pa}(X_i)) \quad (9)$$

This task is commonly referred to as *inferencing*. More precisely, we ask a *conditional probability query*, which can be addressed by various exact as well as approximate inferencing approaches [102]. Variable elimination is a naïve exact inferencing algorithm and is based on a simple idea [102]:

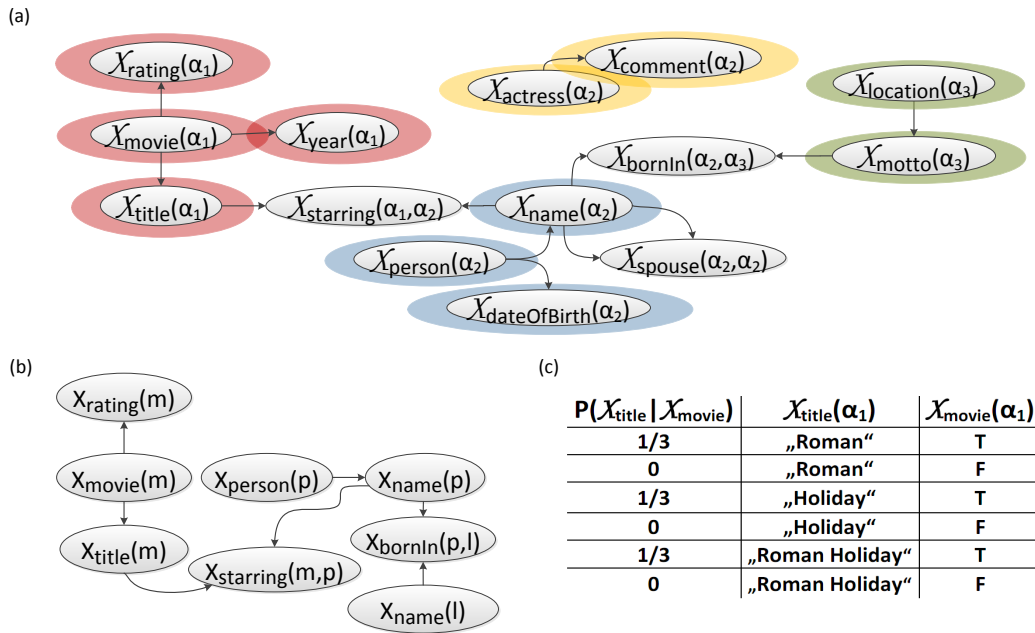


Figure 25: (a) Template-based BN for the running example in Figure 7. (b) Query ground BN for query in Figure 8. Note, templates X_{comment} , X_{motto} and X_{location} are marginalized out. (c) CPD for template X_{title} .

Excursus: Variable Elimination

We know that:

$$P(X_i | \text{pa}) = \frac{P(X_i, \text{pa})}{P(\text{pa})} \quad (10)$$

Thus, to compute the RHS numerator, we can sum out all random variables which are neither the query random variable, X_i , nor the evidence random variables, $\text{Pa}(X_i) = \text{pa}$. The RHS denominator is given by summing out the joint distribution.

Example 26

An example BN (a template-based BN, as discussed below) for our data graph in Figure 7 is illustrated in Figure 25-a. From its structure one can observe that, e.g., X_{starring} is dependent on $\text{Pa}(X_{\text{starring}}) = \{X_{\text{title}}, X_{\text{name}}\}$, but independent of all other random variables given its two parents.

An example CPD for random variable X_{title} is shown in 25-c. Each row captures a probability, given one particular assignment to its parent random variables (Pa). That is, the CPD for X_{title} holds probabilities for n-grams of title values, conditioned on whether or not the particular entity is a Movie.

We use template-based BNs [102] (so-called *template models*) as means to compactly represent correlations in graph-structured data. A template model is a

framework featuring two parts: *template variables* (short: *templates*) and *template factors*.

Each template can be instantiated to obtain multiple random variables in a *ground* BN. These instantiated random variables share the same sample space and the same semantics as their template. More formally, a template is defined as a function $\mathcal{X}(\alpha_1, \dots, \alpha_k)$, whose sample space is $\Omega(\mathcal{X})$ and each argument α_i is a placeholder to be instantiated to obtain random variables.

Example 27

Figure 25-a shows a template model, containing templates such as $\mathcal{X}_{\text{movie}}$, $\mathcal{X}_{\text{starring}}$ or $\mathcal{X}_{\text{name}}$, which are derived from classes, relations, and attributes from our data graph in Figure 7.

A ground BN can be obtained using data from the data graph for template instantiations. That is, placeholders α_i are instantiated by entities in the data, forming an *entity skeleton* of a template.

» Definition 22: Entity Skeleton

Given a data graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \ell_a, \ell_r)$ and a template $\mathcal{X}(\alpha_1, \dots, \alpha_n)$, an entity skeleton of \mathcal{X} is defined as $\mathcal{E}(\alpha_1, \dots, \alpha_n) \subseteq \mathcal{E}(\alpha_1) \times \dots \times \mathcal{E}(\alpha_n)$, where each $\mathcal{E}(\alpha_i) \subseteq \mathcal{V}_E$ in data graph \mathcal{G} specifies all possible entity assignments to α_i .

Using its entity skeletons, we can define a ground BN by instantiating a template as a set of random variables: $\mathbf{X} = \{X(e) \mid e \in \mathcal{E}\}$, where $\Omega(X(e)) = \Omega(\mathcal{X})$.

Example 28

For the template $\mathcal{X}_{\text{person}}(\alpha_2)$ and $\mathcal{E}_{\text{person}}(\alpha_2) = \{p_1, p_2, p_3, p_4\}$, the set of random variables obtained for the ground BN is given by:

$$\mathbf{X}_{\text{person}} = \{X_{\text{person}}(p_1), X_{\text{person}}(p_2), X_{\text{person}}(p_3), X_{\text{person}}(p_4)\}$$

Different assignments to a template argument α_i result in different random variables in the ground BN, which share the same probabilistic semantics. That is, they share the structure dependencies and parameters (CPDs), which are defined in the template model. CPDs are captured as *template factors*, which define probability distributions, which are shared by all instantiated random variables for that particular template.

Such a template-based representation is flexible, since various ground BNs can be obtained, based on different entity skeletons. In our approach, we will exploit this flexibility to define a suitable ground BN for a given query – while relying on a fixed template model.

Example 29

Consider the query ground BN in Figure 25-b. Random variables are instantiated for each query pattern, while having a query variable as placeholder for entity bindings.

4.1.4 Problem

A joint distribution of n random variables that exhibits all possible dependencies requires a high-dimensional representation. A data synopsis (for selectivity estimation), which tries to capture data correlations in such a manner, will suffer from an exponential blowup of storage space and computational cost. At the same time, two random variables are oftentimes actually independent, if conditioned on a third random variable.

Selectivity estimation approaches based on PRM synopses [60, 162] exploit this conditional independence, in order to factor a full joint distribution into multiple low-dimensional distributions. This factorization allows for a compact and efficient synopsis/selectivity estimation.

Unfortunately, Web data and hybrid queries pose novel challenges for selectivity estimation:

Graph-Structured, Schemaless Web Data. Existing PRM-based solutions [60, 162] are proposed for relational data. In particular, they assume a partitioning scheme that determines the tables in which data is stored. Further, such approaches take queries as inputs, which explicitly specify the tables from which data shall be retrieved. For instance, consider the query pattern $\langle x, \text{name}, \text{"Gregory Peck"} \rangle$, which selects all bindings from the entire data graph matching that name. In contrast to that, previous works assume a selection pattern to have a FROM clause that specifies the table from which data is selected, e.g., Person. Thus, the probability $P(X_{\text{name}} = \text{"Gregory Peck"})$ is estimated for bindings in the Person table only. Applying such solutions to a graph-structured setting is not directly possible, because queries do not contain table information. Further, data graphs can be partitioned in various ways. Different partitioning schemes, however, yield different tables, which in turn greatly affect the performance of existing solutions.

Queries over Text-Rich Web Data. Another problem with PRM approaches [60, 162] is that random variables are assumed to have small sample spaces. In existing works, random variables capture structured query patterns with constants that are bounded to a fixed number of values. In addition to structured patterns, we aim to support string patterns for specifying keyword constraints over textual values. In particular, string patterns comprise keywords, which match any value that contains such keywords, see Section 2.1.2. That is, results for these string patterns do not have to exactly match a specified constant, but only have to *contain* a given keyword.

Example 30

For instance, given data in Figure 7 and the query in Figure 8, bindings for $\langle x, \text{name}, \text{"Gregory Peck"} \rangle$ would also satisfy the pattern $\langle x, \text{name}, \text{"Gregory"} \rangle$, as they both contain "Gregory". Thus, to support queries via a string pattern on predicate name, a sample space $\Omega(X_{\text{name}})$ must comprise all words as well as phrases (sequences of words) contained in text values for name. Clearly, $\Omega(X_{\text{name}})$ may potentially be very large.

The problem outlined in the above example is exacerbated, when dependencies between values in these sample spaces have to be considered. For dealing with string patterns, specific *string synopses* summarizing the value spaces of textual attributes have been proposed. For instance, synopses based on pruned suffix trees, Markov tables, clusters or n-grams have received much attention [35, 99, 164]. However, previous works estimate the selectivity of *single* string patterns. In our setting, we aim to support queries that comprise a combination of structured and string query patterns: hybrid queries. *To the best of our knowledge, there is no work, which considers dependencies between these different types of patterns.*

4.2 RESEARCH QUESTIONS AND CONTRIBUTIONS

Next, let us discuss research problems, hypotheses, and contributions, which we present in this chapter.

4.2.1 *Research Questions and Hypotheses*

Coming back to our overall Research Question (introduced in Section 1.3): How to allow for rank-aware and approximate query processing on the Web of data? We aim at this question using rank-aware as well as approximate join operators. In order to integrate such operators in physical query plans, we face another research question:

Research Question 2

How to allow for efficient and effective selectivity estimates on hybrid, schemaless Web data?

Notice, by *efficient* we mean a low selectivity estimation computation time and a small data synopsis size. By *effective* we mean an accurate selectivity estimation.

Throughout this chapter, we address the above research question using hypotheses as follows:

Hypothesis 3

A template-based representation of BNs allows for effective and efficient selectivity estimation for *graph-structured* RDF data.

Concerning Hypothesis 3, we present a template-based BN model, called BN^+ , in Section 4.3, which is well-suited for schemaless, graph-structured Web data.

That is, we expect our BN^+ to enable an effective and efficient data synopsis for selectivity estimation over Web data. We validate this hypothesis in our evaluation on real-world Web data in Section 4.4.

□ **Hypothesis 4**

String synopses can be integrated in template-based BNs and allow for effective and efficient selectivity estimation for *text-rich* RDF data.

With regard to Hypothesis 4, we extend the template-based BN model in Section 4.3 with string synopses. We expect the overall data synopsis to support effective and efficient selectivity estimation of string patterns, which are conjoined with other triple patterns – forming hybrid queries. By means of the evaluation in Section 4.4 we validate this claim.

4.2.2 Contributions

Towards an efficient and effective solution for selectivity estimation of hybrid queries over text-rich Web data, we provide the following contributions:

- *Contribution for Hypothesis 3*

We rely on an instantiation of a general template-based BN: BN^+ . This model is able to effectively capture correlations in schemaless Web data graphs.

In contrast to existing PRM-based approaches [60, 162], we do not assume a specific data partitioning. In fact, our model is learned directly from the instance data – without requiring any schema information.

- *Contribution for Hypothesis 4*

In order to support hybrid queries over text-rich Web data, we show how string synopses can be integrated into our template-based model BN^+ .

This way, we lift the restriction of PRM-based approaches [60, 162] to only feature random variables with small sample spaces. Allowing for large sample spaces is crucial, since selectivity estimation for hybrid queries requires sample spaces to feature all n-grams, which occur in the Web data graph.

- *Contribution for Hypothesis 3 and Hypothesis 4*

We implemented our BN^+ approach to perform experiments on real-world Web data. We can empirically show that our solution greatly improves the effectiveness of selectivity estimates for hybrid queries. In terms of efficiency, our solution is promising, as BN^+ performs comparable to the baseline systems. In fact, the results suggest that BN inferencing requires only a negligible computational overhead.

4.3 SELECTIVITY ESTIMATION OVER TEXT-RICH RDF GRAPHS

In this section, we propose a novel template-based BN, the BN^+ data synopsis, which is well-suited for schemaless, graph-structured Web data. In particular, we show how string synopses can be integrated into BN^+ , in order to support selectivity estimation for hybrid queries. By means of the BN^+ data synopsis, we realize an efficient and effective instantiation of the probabilistic component \mathcal{P} (see Definition 21, p. 72).

4.3.1 BN^+ Data Synopsis

The BN^+ data synopsis comprises two parts: (1) A template-based BN, which is defined based on instances (entity skeletons) observed in the Web data graph. (2) String synopses to summarize the sample spaces for predicates in the data graph, which feature large textual values.

4.3.1.1 BN^+ Template Model


Given a Web data graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \ell_a, \ell_r)$, we introduce three kinds of templates:

- *Attribute Template*
We define a template for each attribute $a \in \ell_a$ in data graph \mathcal{G} : $\mathcal{X}_a(\alpha_1)$.
- *Relation Template*
We define a template for each relation $r \in \ell_r$ in data graph \mathcal{G} : $\mathcal{X}_r(\alpha_1, \alpha_2)$.
- *Class Template*
We define a template for each class $c \in \mathcal{V}_C$ in data graph \mathcal{G} : $\mathcal{X}_c(\alpha_1)$.

Each template for a relation r or a class c is binary:

$$\Omega(\mathcal{X}_r) = \Omega(\mathcal{X}_c) = \{\mathbf{T}, \mathbf{F}\}$$

In contrast, a sample space for attribute template \mathcal{X}_a comprises a bag of n -grams, which are derived from values of attribute a .

 **Example 31**

For our running example, templates are depicted in Figure 25-a. For instance, the sample space for $\mathcal{X}_{\text{name}}$ is given by:

$$\Omega(\mathcal{X}_{\text{name}}) = \{“Audrey”, “Hepburn”, “Audrey Hepburn”, “Mel”, \dots\}$$

To obtain a ground BN, above templates are instantiated using the following entity skeletons:

- *Entity Skeleton for Attribute Templates*
For an attribute template, an entity skeleton consists of all entities having that attribute:

$$\mathcal{E}_a(\alpha_1) := \{s \mid \langle s, a, o \rangle \in \mathcal{E}_A\} \tag{11}$$

- *Entity Skeleton for Relation Templates*

The entity skeleton $\mathcal{E}_r(\alpha_1, \alpha_2)$ contains all pairs of source/target entities having relation r :

$$\mathcal{E}_r(\alpha_1, \alpha_2) := \{\langle s, t \rangle \mid \langle s, r, t \rangle \in \mathcal{E}_R\} \quad (12a)$$

Let source and target entities be denoted as

$$\mathcal{E}_r^s(\alpha_1) := \{s \mid \langle s, r, t \rangle \in \mathcal{E}_R\} \quad (12b)$$

and

$$\mathcal{E}_r^t(\alpha_2) := \{t \mid \langle s, r, t \rangle \in \mathcal{E}_R\} \quad (12c)$$

where

$$\mathcal{E}_r(\alpha_1, \alpha_2) \subseteq \mathcal{E}_r^s(\alpha_1) \times \mathcal{E}_r^t(\alpha_2) \quad (12d)$$

- *Entity Skeleton for Class Templates*

The entity skeleton for a class template $\mathcal{X}_c(\alpha_1)$ is given by all entities belonging to that class:

$$\mathcal{E}_c(\alpha_1) := \{s \mid \langle s, \text{type}, c \rangle \in \mathcal{E}\} \quad (13)$$

Such a template-based approach has the merit of being compact. The number of templates is far less than the number of random variables in a ground BN. This is because BN structure and BN parameters (CPDs) are only learned for the template model. More precisely, templates are instantiated with entities at runtime to construct a ground BN. For inferencing, a CPD from the template model is shared among all random variables in the ground BN, which instantiate that template.

4.3.1.2 Discussion

In our work, we use a general template-based model as probabilistic framework, see problem description in Section 4.1.4. Previous instantiations of template-based models focus on relational data. Most notably, Probabilistic Relational Models (PRM) [57] and Probabilistic Entity Relation Models (PER) [59] have been proposed. In fact, PRMs have also been applied for selectivity estimation [60, 162]. However, PRM-based solutions are not well-suited for a graph-structured data model, because of differences in the data as well as query model.

In a relational context, data is stored in tables corresponding to relations captured by a conceptual model. Further, relation names are explicitly given in a query – stated in a FROM clause. Correspondingly, previous works [60, 162] employ a PRM to model selection predicates through random variables of the form $X_{R,A}$, where R is a relational table and A is an attribute. For instance, $X_{\text{Person.name}} = \text{“Audrey”}$ is a random variable capturing a selection on table `Person` where `name` equals `“Audrey”`. Analogously, join predicates are modeled as binary random variables that involve two explicitly specified tables.

As opposed to that, graph-structured data, such as RDF, can be partitioned in different ways. For instance, there may be a table for every entity class, e.g., a Person table capturing different person attributes [168]. On the other hand, a table might be constructed for every attribute and relation – leading to, e.g., a table for attribute name. The latter partitioning is also known as *vertical partitioning* [9]. Thus, at query level, there is no explicit information about the tables from which data shall be selected. Further, schema information may be queried via class patterns, which are not supported in the relational setting.

Due to these differences, the following problems occur when storing graph data in tables and applying a PRM-based approach:

- *Sensitivity to Data Partitioning*

A PRM assumes tables to be given. Thus, random variables are defined and their parameters/dependencies are learned – all of which with regard to these tables. Different partitioning schemes for data graphs, however, yield different tables. Therefore, models learned from such tables might largely vary – in terms of dependency structure as well as parameters. In particular, [162] focuses on learning correlations between attributes, which are comprised within one table, while assumptions are made to simplify cross-table dependencies. While resulting in a very lightweight PRM, this approach assumes that data is partitioned in tables comprising related attributes. In the case of vertical partitioning, however, where every attribute constitutes a table, there are no local dependencies to be learned and cross-table dependencies are more important. Generally speaking, the performance of PRM solutions is sensitive to the partitioning strategy. Our template-based solution does not make any assumptions about data partitioning. Instead, a template model is learned from entity skeletons and values from a data graph – independent from the way data is stored in tables.

- *Cross-Table Selection*

Besides vertical partitioning, another common strategy for graph data partitioning is to construct a table for every class [168]. However, oftentimes common attributes, such as name, are used to describe entities of different types, e.g., Person and Location (see Figure 7). Given such a class-based partitioning, the attribute pattern $\langle p, \text{name}, \text{"Audrey"} \rangle$ would select data from different tables. Unfortunately, these tables may not be explicitly specified in a query. At the same time, this explicit specification is required by PRM-based approaches. A possible solution is to maintain information to find out in which tables name occurs and to construct corresponding random variables, which refer to these different tables. Finally, one would need to aggregate the probabilities obtained for these variables. In contrast, with our template-based solution, only one template variable, X_{name} , is needed to support this predicate. A PRM-based approach, on the other hand, requires consulting one variable and CPD for every table.

- *Multi-Table Joins*

A similar problem occurs when dealing with joins. In a PRM context, a join predicate requires data from two explicitly specified tables. Joins correspond to relation patterns in our setting. That is, a relation may be seen as referring to two foreign keys, which connect a source with a target entity. However, depending on the data partitioning, processing such a join (relation pattern), might involve one or more unspecified tables. For instance, given a relation pattern $\langle p, \text{bornIn}, l \rangle$, predicate `bornIn` could join either `Person` or `Actress` entities with `Location` instances, see Figure 7. Thus, data for the entities to be joined might be located in different tables. In a PRM one may handle this issue via using several random variables and aggregating their probabilities. In contrast, using our approach, merely one single CPD and random variable representing the given relation pattern is required.

4.3.1.3 BN^+ String Synopsis

Consider an attribute template \mathcal{X}_a with sample space $\Omega(\mathcal{X}_a)$. Then, for computing selectivity estimates for string patterns over attribute a , the sample space $\Omega(\mathcal{X}_a)$ must capture all words and phrases, which occur in attribute a 's values. Oftentimes attribute values comprise long texts, resulting in a sample space to quickly blow up. So, we propose to employ string synopses in order to compactly represent large sample spaces of attribute templates.

More precisely, in order to compactly represent Ω_a , which is a large set of strings, we propose the use of string synopses such as Markov tables [35], histograms [99], or n -gram synopses [164]. We generalize from existing works to define the following class of string synopses:

» **Definition 23: String Synopsis**

A string synopsis for an attribute template \mathcal{X}_a is a tuple $\mathcal{S}(\nu, \text{count})$.

- The synopsis function ν maps elements in the bag of n -grams for attribute a , denoted by \mathcal{B}_a , to elements in a compact synopsis sample space $\Omega(\mathcal{X}_a)$.
- A function $\text{count} : \Omega(\mathcal{X}_a) \rightarrow \mathbb{N}$ returns the number of elements in the “original” space \mathcal{B}_a , which are represented by a given synopsis element in $\Omega(\mathcal{X}_a)$.


Above definition of a synopsis is generic, however, a well-suited synopsis function ν should aim at three goals:

- ① The synopsis function ν should lead to a small sample space, $\Omega(\mathcal{X}_a)$, since a compact representation facilitates learning and keeps the CPD size small.
- ② The synopsis function ν should be most accurate. That is, each synopsis element in $\Omega(\mathcal{X}_a)$ should represent only few n -grams from the original space, \mathcal{B}_a .

- ③ The synopsis function ν should capture all “important” n-grams, while discarding “not important” ones. From a conceptual point of view, discarded n-grams are mapped to a bottom element \perp , which captures the probability mass of all missed n-grams.

While we do not restrict our approach to a particular type of string synopsis, existing work [164] has shown that (with regard to above goals) synopses based on n-grams are well-suited for the task of selectivity estimation for the *contains operator on dictionaries*. Notice, the contains operator has the same semantics as our string patterns (see Definition 9, p. 23): it matches text values that contain a given keyword. Thus, we follow this line of work and integrate n-gram synopses into BN^+ for our evaluation systems in Section 4.4.


An n-gram synopsis works as follows [164]: For an attribute a in the data graph, the synopsis function ν projects a 's textual attribute values to their n-gram representation \mathcal{B}_a .

 **Example 32**

The attribute `comment` has one attribute value in Figure 7. This value is mapped by the synopsis function ν to $\mathcal{B}_{\text{comment}} = \{\text{“Audrey”, “Hepburn”, “Audrey Hepburn”, ...}\}$.

Then, the space \mathcal{B}_a is reduced by using a decision criterion to dictate which n-grams in \mathcal{B}_a to include in a synopsis sample space $\Omega(\mathcal{X}_a)$. That is, the synopsis space $\Omega(\mathcal{X}_a)$ represents a subset of “important” n-grams. Note, n-gram synopses are most accurate, since each synopsis element in $\Omega(\mathcal{X}_a)$ represents exactly one n-gram in \mathcal{B}_a – in contrast to, e.g., histograms.

Recent work has outlined several of such decision criteria [164]. One simplistic strategy is to *randomly sample* n-grams from \mathcal{B}_a . Another approach is to construct a *top-k* n-gram synopsis. For this, n-grams are extracted together with their number of occurrences. Then, the k most frequent n-grams are included in the synopsis space.

 **Example 33**

Let us continue Example 32. Given attribute `comment`, the count for n-gram “Audrey” would be two, while “Hepburn” only occurs once. Thus, the top- k n-gram synopsis would rank “Audrey” as more important than “Hepburn”. In other words, the synopsis would rather discard the n-gram “Hepburn” than the n-gram “Audrey”.

Further, as a more efficient top- k n-gram synopsis, a *stratified Bloom filter* synopsis has been proposed [164]. This synopsis uses Bloom filters [27] as a heuristic map that projects n-grams to their counts. This way, the stratified Bloom filter synopsis can store more n-grams than the top- k n-gram synopsis. However, this extended storage capacity comes at the cost of accuracy, since Bloom filters resemble a probabilistic data structure.

4.3.2 BN^+ Data Synopsis Construction

The BN^+ data synopsis should *compactly represent the joint distribution over templates*, while capturing dependencies between structured data elements as well as n-grams of textual attribute values.

However, large sample spaces and complex dependencies among templates may lead to prohibitive synopsis size/selectivity estimation times. In fact, during our experiments we observed sample space sizes up to 2 million n-grams for some attributes. Such sample spaces translate to large CPDs, which in turn make fast selectivity estimation at runtime impossible. Furthermore, dependencies between templates aggravate this problem: the size of a CPD multiplies with each parent a particular template is dependent on. We address these problems by means of two strategies:

① *Compact Sample Spaces*

We utilize string synopses in order to compress an attribute template sample space into a manageable size.

② *Compact BN Structure*

Instead of constructing a complex network structure featuring all possible dependencies, we solely focus on the most important ones. That is, we aim for an approximation of the joint distribution that shall limit the dimensions of the CPDs, while preserving key dependencies.

In the following paragraphs, we will show in Algorithm 5 how string synopses (as presented in the previous Section 4.3.1.3) can be integrated in the BN^+ synopsis. Moreover, we introduce an structure learning approach, which solely captures key dependencies in the data graph. Last, we discuss how BN parameters can be learned.

4.3.2.1 Structure Learning

An efficient and well-known technique in the BN literature [41, 119] is based on using a *product approximation* of rich structures via trees. These tree structures guarantee that each template has at most one parent.

Recently, such an approximation has been adopted to PRMs for a relational setting. The resulting “lightweight” structure has been shown to improve efficiency, while still producing high-quality selectivity estimates [162].

We apply product approximation to a graph-structured setting, by imposing a *fixed structure of independences between template variables*:

» **Definition 24: BN^+ Fixed Network Structure**

Given a data graph, the following conditional independences are assumed to hold:

- a. Two templates \mathcal{X}_1 and \mathcal{X}_2 are conditionally independent given their parents, if they do not share a common entity in their skeletons \mathcal{E}_1 and \mathcal{E}_2 .

Note, in case either of these templates, say \mathcal{X}_i , captures a relation, we use $\mathcal{E}_i = \mathcal{E}_i^s \cup \mathcal{E}_i^t$ as the skeleton, i.e., the union of its source and target entities

- b. Each class template, \mathcal{X}_c , has no parent.
- c. Each relation template, \mathcal{X}_r , is independent of any class template \mathcal{X}_c , given its parents.

We argue that the independences induced via the above fixed structure are meaningful due to the following argumentation:

- ① We impose that strong correlations among templates only occur, if they share some common entities. Intuitively speaking, for templates to be correlated, they need to “talk about” the same entities, see Definition 24-a.
- ② We argue that there is a causal dependency between a class template and an attribute template (see Definition 24-b/c). In other words, assigning an entity to a given class causally affects the probability of its attribute values.
- ③ Last, we impose a conditional independence between a class and a relation template (see Definition 24-b/c). That is, we assume that class templates influence attribute templates, which, in turn, influence relation templates. This way, we have a lightweight “dependency chain”, which starts with the class templates.

Exploiting the fixed structure, we can decompose the structure learning procedure: First, we construct a disconnected graph, coined *local part* and denoted as $\mathcal{T}_{\text{local}}$, of the template model by learning dependencies between class/attribute and attribute/attribute template pairs. Then, we simplify $\mathcal{T}_{\text{local}}$ via an approximation $\mathcal{T}_{\text{local}}^*$. Last, we add relation templates to the structure $\mathcal{T}_{\text{local}}^*$ and obtain a final template model \mathcal{T} .

For learning the local part, $\mathcal{T}_{\text{local}}$, we add weighted edges between each class/attribute and attribute/attribute template pair, which is not independent with respect to the fixed structure assumption in Definition 24. That is, each pair must have an “overlap” in their skeletons – the templates share one or more common entities.

Example 34

In Figure 25-a, we add an edge $\mathcal{X}_{\text{movie}} \rightarrow \mathcal{X}_{\text{title}}$ for the Movie/title template pair, because their skeletons are identical:

$$\mathcal{E}_{\text{movie}} = \mathcal{E}_{\text{title}} = \{m_1\}$$

In order to calculate the dependency weight between two templates, we use the *mutual information* quantity, denoted as mi , which represents the “amount of information shared” between two templates $\mathcal{X}_1, \mathcal{X}_2$:

$$mi(\mathcal{X}_1, \mathcal{X}_2) := \sum_{x_1 \in \Omega_1} \sum_{x_2 \in \Omega_2} P(x_1, x_2) \cdot \log \left(\frac{P(x_1, x_2)}{P(x_1)P(x_2)} \right) \quad (14a)$$

with $\Omega_i = \Omega(\mathcal{X}_i)$ being the sample space of template \mathcal{X}_i . The maximum likelihood estimation of $P(\mathcal{X}_i = x_i)$ is:

$$P(\mathcal{X}_i = x_i) := \frac{\check{M}_i[x_i]}{N} \quad (14b)$$

with N as normalization factor. \check{M} is a sufficient statistic that counts entities in the skeleton of \mathcal{X}_i having x_i as value:

$$\check{M}_i[x_i] := \sum_{e \in \mathcal{E}_i} \mathbb{1}\{\mathcal{X}_i(e) = x_i\} \quad (14c)$$

Note, $\mathbb{1}$ is an indicator function, i.e., it returns 1 if its expression is true, 0 otherwise. Similarly, the joint distribution of \mathcal{X}_1 and \mathcal{X}_2 is:

$$P(\mathcal{X}_1 = x_1, \mathcal{X}_2 = x_2) := \frac{\check{M}_{1,2}[x_1, x_2]}{N} \quad (14d)$$

with N as normalization factor and $\check{M}_{1,2}[x_1, x_2]$ as count of entities having both values:

$$\check{M}_{1,2}[x_1, x_2] := \sum_{e \in \mathcal{E}_1 \cap \mathcal{E}_2} \mathbb{1}\{\mathcal{X}_1(e) = x_1, \mathcal{X}_2(e) = x_2\} \quad (14e)$$

Once the weighted edges have been added to the local parts $\mathcal{T}_{\text{local}}$, the model comprises all possible dependencies between class templates and attribute templates according to our fixed structure in Definition 24. Then, we capture only the most important correlations in $\mathcal{T}_{\text{local}}$ by reducing it to its *maximum-spanning forest*.³⁰ This yields a much simpler structure: $\mathcal{T}_{\text{local}}^*$.

Example 35

For our running example, $\mathcal{T}_{\text{local}}^*$ is depicted in Figure 25-a and its four maximum-spanning trees are highlighted in different colors. For instance, the red maximum-spanning tree contains three edges (dependencies):

$$\{\mathcal{X}_{\text{movie}} \rightarrow \mathcal{X}_{\text{title}}, \mathcal{X}_{\text{movie}} \rightarrow \mathcal{X}_{\text{year}}, \mathcal{X}_{\text{movie}} \rightarrow \mathcal{X}_{\text{rating}}\}$$

Intuitively speaking, each maximum-spanning tree describes dependencies of the same entities – as dedicated by Definition 24-a. For example, the above maximum-spanning tree captures dependencies of Movie entities.

Due to the fixed structure restriction, the maximum-spanning forest algorithm may find no solution. In such cases, we iteratively remove the weakest attribute-to-attribute edge, until a spanning tree can be obtained.

³⁰In this work, a maximum-spanning forest is defined as a set of spanning trees – one for each component in $\mathcal{T}_{\text{local}}$ [148].

Algorithmus 5 : Construction of the BN^+ data synopsis.


Input : Templates $\mathcal{X} = \{X_a\}_{\forall a \in \ell_a} \uplus \{X_r\}_{\forall r \in \ell_r} \uplus \{X_c\}_{\forall c \in \mathcal{V}_c}$,
entity skeletons $\mathcal{E} = \{\mathcal{E}_a\}_{\forall a \in \ell_a} \uplus \{\mathcal{E}_r\}_{\forall r \in \ell_r} \uplus \{\mathcal{E}_c\}_{\forall c \in \mathcal{V}_c}$,
and synopsis size ρ .

Output : BN^+ data synopsis \mathcal{T} .

```

1 begin
2    $\mathcal{T}_{\text{local}} \leftarrow \emptyset$ 
3    $\Omega(X_r) = \Omega(X_c) = \{\mathbf{T}, \mathbf{F}\}$  for all  $c$  and  $r$ 
4    $\Omega(X_a) = \text{InitializeSynopsis}(a, \rho)$  for all  $a$ 
5   foreach  $X_a \in \mathcal{X}$  do
6     foreach non-independent  $X_c$  to  $X_a$  w.r.t. Definition 24 do
7       Add  $(X_c \xrightarrow{\text{mi}(X_c, X_a)} X_a)$  to  $\mathcal{T}_{\text{local}}$ 
8     foreach non-independent  $X_{a'}$  to  $X_a$  w.r.t. Definition 24 do
9       Add  $(X_{a'} \xrightarrow{\text{mi}(X_{a'}, X_a)} X_a)$  to  $\mathcal{T}_{\text{local}}$ 
10     $\mathcal{T}_{\text{local}}^* \leftarrow \text{MAX-SPANNING-FOREST}(\mathcal{T}_{\text{local}})$ 
11     $\mathcal{T} \leftarrow \mathcal{T}_{\text{local}}^*$  // initialize  $\mathcal{T}$ 
12    foreach  $X_r \in \mathcal{X}$  do
13       $X_{a_s}^{\text{best}} = \arg \max_{a \in \text{source}(r)} \text{mi}(X_a, X_r)$ 
14       $X_{a_t}^{\text{best}} = \arg \max_{a \in \text{target}(r)} \text{mi}(X_a, X_r)$ 
15      if  $X_{a_s}^{\text{best}} \neq \text{NULL}$  then
16        Add  $(X_{a_s}^{\text{best}} \rightarrow X_r)$  to  $\mathcal{T}$ 
17      if  $X_{a_t}^{\text{best}} \neq \text{NULL}$  then
18        Add  $(X_{a_t}^{\text{best}} \rightarrow X_r)$  to  $\mathcal{T}$ 
19  return  $\mathcal{T}$ 

```

 **Example 36**

Continuing Example 35: While the orange and blue maximum-spanning trees have been comprised in one component in $\mathcal{T}_{\text{local}}$, we needed to remove the weighted edges between X_{name} and X_{comment} , which led to two trees in the maximum-spanning forest $\mathcal{T}_{\text{local}}^*$, see Figure 25-a.

Overall, the construction of $\mathcal{T}_{\text{local}}^$ results in a dynamic partitioning of the dependencies, based on information contained in entity skeletons.*

Next, we integrate relation templates in the maximum-spanning forest $\mathcal{T}_{\text{local}}^*$. Mutual information is used to quantify dependencies between relation templates and attribute templates. For every relation template, its mutual information with regard to all possible (with regard to Definition 24, p. 84) source and target attribute templates is computed. Finally, given a relation template, the two attribute templates that exhibit the highest mutual information are used as parents of that relation template.

Example 37

In Figure 25-a, the relation template $\mathcal{X}_{\text{starring}}$ connects two maximum-spanning trees (red and blue) from $\mathcal{T}_{\text{local}}^*$ via two attribute templates: $\mathcal{X}_{\text{title}}$ and $\mathcal{X}_{\text{name}}$. More specifically, relation template $\mathcal{X}_{\text{starring}}$ has two parents, $\mathcal{X}_{\text{title}}$ and $\mathcal{X}_{\text{name}}$, which exhibit the highest mutual information among all possible parent templates.

Algorithm 5 summarizes the entire structure learning procedure. The algorithm takes all relation/attribute/class templates, their entity skeletons, and the string synopsis size parameter ρ as input. Then, we initialize the sample spaces of all templates in Lines 3-4.

In particular, initializing the sample space for an attribute template requires string synopsis construction for the associated attribute. Note, the string synopsis construction algorithm depends on the employed string synopsis. We discussed the construction of n-gram string synopses in Section 4.3.1.3.

We learn the local template model $\mathcal{T}_{\text{local}}$ on Lines 5-9, by adding weighted edges between all possible attribute/attribute and class/attribute template pairs. On Line 10, we approximate $\mathcal{T}_{\text{local}}$ by computing the maximum-spanning forest and adding its result, $\mathcal{T}_{\text{local}}^*$, to an intermediate model. Relation templates are added as connections between maximum-spanning trees in $\mathcal{T}_{\text{local}}^*$ on Lines 12-18 – resulting in the BN^+ synopsis, \mathcal{T} .

Constructing a BN^+ data synopsis by means of Algorithm 5 leads to a *valid* template-based BN :

• Theorem 4

The template-based BN^+ synopsis constructed according to Algorithm 5 is valid, i.e., acyclic.

Sketch of Proof

A local model $\mathcal{T}_{\text{local}}$ is reduced to a forest of trees, $\mathcal{T}_{\text{local}}^*$, via a maximal spanning tree algorithm. Thus, every tree in $\mathcal{T}_{\text{local}}^*$ represents a valid acyclic fragment of $\mathcal{T}_{\text{local}}$. Then, we connect these tree structures by incrementally adding edges representing relation templates, see Algorithm 5 on Lines 12-18. However, a relation template must not have children. Thus, no cycles can be introduced at this step ■

4.3.2.2 Parameter Learning

After having built a BN^+ network structure, we may now learn its model parameters, i.e., conditional probability distributions. As done in recent works [60, 162], learning CPDs can be achieved based on the sufficient statistic \check{M} in Equation 14. More precisely, according to Bayes rule it holds that:

$$P(\mathcal{X}_i | \mathcal{X}_j) = \frac{P(\mathcal{X}_i, \mathcal{X}_j)}{P(\mathcal{X}_j)} \quad (15)$$

So, we can compute $P(\mathcal{X}_i, \mathcal{X}_j)$ and $P(\mathcal{X}_j)$, as we did for obtaining the mutual information in Equation 14a. Note, in case of a relation template, say \mathcal{X}_i , we need to estimate a distribution conditioned on two other templates (\mathcal{X}_j and \mathcal{X}_k):

$$P(\mathcal{X}_i | \mathcal{X}_j, \mathcal{X}_k)$$

This can be achieved by extending the \check{M} function to capture three templates: $\check{M}_{1,2,3}[x_1, x_2, x_3]$. For an efficient parameter learning, we employ two sorts of optimizations.

- ① We use caching strategies for keeping frequently needed \check{M} statistics in memory. In fact, caching can be applied to store results already produced during structure learning.

Example 38

For example, sufficient statistics for the template $\mathcal{X}_{\text{Movie}}$ are needed more than once, because $\mathcal{X}_{\text{Movie}}$ is a parent of $\mathcal{X}_{\text{title}}$, $\mathcal{X}_{\text{year}}$, and $\mathcal{X}_{\text{rating}}$. So, we can cache sufficient statistics for $\mathcal{X}_{\text{Movie}}$, thereby omitting additional computations.

- ② We can formulate \check{M} expressions (see Equation 14) as queries to be issued at a database. For instance, $\check{M}_{p_1, p_2}[x_1, x_2]$ can be calculated based on the cardinality of a result set for query $\mathcal{Q} = \{\langle s, p_1, x_1 \rangle, \langle s, p_2, x_2 \rangle\}$, with \mathcal{X}_{p_i} being the template for p_i . This way, the database handles query optimizations, caching of results for frequently computed query fragments etc.

4.3.2.3 Maintenance

Data on the Web is subject to frequent changes. We handle these evolving triples in two ways:

On the one hand, changes may result in minor modifications of entity skeletons and sample spaces. As a consequence, some model parameters may no longer be accurate enough for effective selectivity estimations. Such affected CPDs should be recomputed, given an updated data graph. For minor changes such a reestimation, however, does not influence other parameters and/or the structure. So, these computations may be done incrementally. In fact, while model parameters might have to be frequently recomputed, the network structure is commonly much more “stable”.

On the other hand, given drastic changes in a data graph, its structure as well as parameters has to be recomputed. Our experiments show that, even in this case, learning is feasible within a short amount of time. For our datasets, we observed that computation of the entire BN^+ model, including string synopses, took at most 3 hours.

4.3.3 Selectivity Estimation

4.3.3.1 Query Ground BN

In order to employ our BN^+ data synopsis for selectivity estimation, we have to *instantiate its templates specifically for the given query – leading to ground BN*.

To be precise, we do not form a standard ground BN, since this would solely capture entities as random variable assignments. Instead, we form a query ground BN featuring random variables that have *sets* of entities as assignments, which are result bindings to query variables.

We instantiate templates in the BN^+ data synopsis as follows:

- For each relation pattern $\langle s, r, o \rangle$ and class pattern $\langle s, \text{type}, c \rangle$ in \mathcal{Q} , we instantiate a random variable $X_r(s, o) = \mathbf{T}$ and $X_c(s) = \mathbf{T}$, respectively.
- For every string pattern $\langle s, a, w \rangle$, its keyword w is mapped to a corresponding element in our synopsis, $v(w)$, such that the resulting instantiated random variable is: $X_a(s) = v(w)$.
- Last, for an attribute pattern $\langle s, a, o \rangle$ we instantiate the random variable: $X_a(s) = \mathbf{o}$.

It is important to note that any template in the BN^+ synopsis, which is not needed for a given query, is marginalized out. Moreover, the same template may be instantiated multiple times – as required for the query.

Example 39

In our running example, we instantiate one random variable for each query pattern, as shown in Figure 25-b. In particular, we need two instantiations of the template $\mathcal{X}_{\text{name}}$, since the query has two triple patterns with predicate name. Templates that are not relevant for the query, e.g., $\mathcal{X}_{\text{comment}}$, are marginalized out.

Given a query \mathcal{Q} with query graph $\mathcal{G}^{\mathcal{Q}} = (\mathcal{V}^{\mathcal{Q}}, \mathcal{E}^{\mathcal{Q}})$, we compute the joint probability of the associated query ground BN to estimate \mathcal{Q} 's selectivity:

$$P(\mathcal{Q}) \approx \gamma \cdot P \left(\begin{array}{cc} \bigwedge_{\substack{\langle s, r, o \rangle \in \mathcal{E}^{\mathcal{Q}} \\ r \in \ell_r}} X_r(s, o) = \mathbf{T} & \bigwedge_{\substack{\langle s, \text{type}, c \rangle \in \mathcal{E}^{\mathcal{Q}} \\ c \in \mathcal{V}_c}} X_c(s) = \mathbf{T} \\ \bigwedge_{\substack{\langle s, a, w \rangle \in \mathcal{E}^{\mathcal{Q}} \\ a \in \ell_a, w \in \mathcal{V}_k^{\mathcal{Q}}}} X_a(s) = v(w) & \bigwedge_{\substack{\langle s, a, o \rangle \in \mathcal{E}^{\mathcal{Q}} \\ a \in \ell_a, o \in \mathcal{V}_c^{\mathcal{Q}}}} X_a(s) = \mathbf{o} \end{array} \right) \quad (16a)$$

where

$$\gamma := \frac{1}{\prod_{w \in \mathcal{V}_k^{\mathcal{Q}}} \text{count}(v(w))} \quad \text{is a correction factor.} \quad (16b)$$

Example 40

Coming back to Example 39, we compute $P(Q)$ as:

$$\begin{aligned} P(Q) &\approx \gamma \cdot P(X_{\text{Movie}}(m) = \mathbf{T} \wedge X_{\text{Rating}}(m) = 8.0 \wedge \\ &\quad X_{\text{Title}}(m) = v(\text{"Holiday"}) \wedge X_{\text{Starring}}(m, p) = \mathbf{T} \wedge \\ &\quad X_{\text{Person}}(p) = \mathbf{T} \wedge X_{\text{Name}}(p) = v(\text{"Audrey Hepburn"}) \wedge \\ &\quad X_{\text{BornIn}}(p, l) = \mathbf{T} \wedge X_{\text{Name}}(l) = v(\text{"Belgium"})) \end{aligned}$$

The correction factor γ is necessary, because $v(w)$ may not only capture the probability mass for keyword w , but could also include other words (phrases). Consider a histogram synopsis. Here, “Wiliam” and “Wylar” may be represented by a single bucket $[W_i - W_y]$. Then, a query pattern $\langle p, \text{name}, \text{"Wiliam"} \rangle$ would be translated to $X_{\text{name}}(p) = [W_i - W_y]$. However, the bucket $[W_i - W_y]$ not only comprises “Wiliam”, but also “Wylar”. Thus, its probability must be “corrected”. Note, such a correction implies a uniform distribution among all words (phrases), which are captured by a single synopsis element.

For the above inferencing problem (Equation 16), each instantiated random variable reuses the CPD from its template. In the simplest case, inferencing for $P(Q)$ could be performed via “brute-force” marginalization. However, as marginalization is expensive, we employ belief propagation allowing an approximation, which operates on a *junction tree* representation of the ground BN [162].

Further, we adopt the standard inferencing task to deal with the following problems that arise in our setting: *multiple value assignments* and *missing synopsis values*.


4.3.3.2 Multiple Value Assignments

Often times a string synopsis restricts the length of its phrases due to a limited amount of storage space. If a query pattern contains a phrase as keyword, which is longer than this threshold, a simple strategy is to break that phrase into multiple smaller phrases. For instance, if a synopsis only allows 1-grams, a keyword phrase with k words must be split into k 1-grams. In such a case, instantiated random variables (referring to the same query variable) have multiple values.

Example 41

Let’s assume we have an n -gram synopsis, which allows only 1-grams. Then, the random variable $X_{\text{name}}(p)$ in Figure 25-b would have *two* assignments (“Audrey” and “Hepburn”), because the query keyword “Audrey Hepburn” is too long.

This problem can be addressed through an *aggregation function*. We use a *stochastic mode* aggregation, which uses all values as evidence, but weights each one with its frequency within the query [155].

 **Example 42**

Continuing the above Example 41, $P(X_{\text{bornIn}} = \mathbf{T} \mid X_{\text{name}}(p) = \text{“Audrey Hepburn”})$ is computed via stochastic mode aggregation as:

$$\begin{aligned} P(X_{\text{bornIn}} = \mathbf{T} \mid X_{\text{name}}(p) = \text{“Audrey Hepburn”}) \approx \\ \frac{1}{2} \cdot P(X_{\text{bornIn}} = \mathbf{T} \mid X_{\text{name}}(p) = \text{“Audrey”}) + \\ \frac{1}{2} \cdot P(X_{\text{bornIn}} = \mathbf{T} \mid X_{\text{name}}(p) = \text{“Hepburn”}) \end{aligned}$$

This is because, the probability for each assignment of $X_{\text{name}}(p)$ is weighted with $\frac{1}{2}$, since both values (“Audrey” and “Hepburn”) occur once in the query.

4.3.3.3 Missing Synopsis Values

There are synopses, such as the top-k n-gram, for which some query keywords do not have a corresponding synopsis element. That is, the synopsis discarded that particular word (phrase) during construction for space reasons. The probability for these “missing” keywords cannot be estimated by means of the BN^+ synopsis, since such keywords are not included in any sample space. To deal with this problem, a string pattern featuring a missing keywords is assumed to be *independent from the remainder of the query*.

Then, its probability can be estimated based on a heuristic. We employ the left-backoff strategy, which finds the longest known n-gram that is a prefix (postfix) of the missing keyword and estimates its probability based on statistics for that prefix (postfix) [164].

4.4 EVALUATION

In the following, we discuss the experiments, which we preformed to analyze the *accuracy* (effectiveness) and the *time performance* (efficiency) of our BN^+ selectivity estimation. As baseline, we used an approach that assumes independence among string patterns as well as between them and structured query patterns.

Overall, our results suggest that the baseline yields very low accuracy, when dependencies between query patterns exist. For IMDB, we observed such strong correlations in the data. Here, given we employ the most accurate string synopsis (stratified bloom filters), BN^+ improved the baseline’s accuracy by 93% in terms of multiplicative error. In other words, BN^+ achieved a decrease of error by a factor of 13.6. With respect to efficiency, we found that the BN inferencing overhead was actually negligible. The main factor driving computation time was the string synopsis that we employed. When both approaches, BN^+ and the baseline, used the same type of string synopsis, their performance was comparable.

	IMDB	DBLP
# Triples	7,310,190	11,014,618
# Entities	1,673,097	2,395,467
# 1-grams	7,841,347	25,540,172
# Attributes	11	21
# Relations	8	18
# Classes	6	18

Table 1: Dataset statistics for DBLP and IMDB benchmarks.

	# Relation Patterns			# String Patterns		
	0	1	2-4	1-2	3	4-7
# Queries	33	44	23	28	35	26
	# Class Patterns			# Total Patterns		
	1	2	3-4	2-3	4-6	7-11
# Queries	49	30	21	28	31	41

Table 2: Query statistics depicting the number of relation patterns, string patterns, and class patterns, which are contained in our query load.

4.4.1 Evaluation Setting

Data. We used two real-world datasets: DBLP comprising computer science bibliographies and IMDB holding information from the movie domain. Table 1 provides basic statistics for both datasets. DBLP as well as IMDB hold text-rich attributes like name, label, or info.

We employed n-gram string synopses as presented in [164]. However, we only used 1-grams in our experiments, as larger values for n resulted in synopses that exceed our memory space limit. Overall, we extracted 25,540,172 and 7,841,347 1-grams from DBLP and IMDB.

We chose these two datasets, as in one of them (IMDB) textual attribute values strongly correlate among each other and with structured data. In particular, we noticed strong dependencies during structure learning between values of attributes such as label and info. Hence, IMDB is appropriate to test the hypothesis: *assuming independence hurts the quality of selectivity estimates, given a dataset exhibits correlations*. On the other hand, we employed DBLP, which showed almost no such correlations. Here, we expect accuracy differences to be less significant. Comparing the accuracy performances across these two datasets will illustrate the *relative* benefit of our solution.

Queries. We employed queries that have been used for keyword search evaluation [42, 114]. These queries capture information needs expressed as keywords.

Based on query keywords and their structured results, we constructed corresponding graph patterns, comprising string, class, and relation patterns. In particular, we generated 54 DBLP queries based on [114]. Additionally, 46 queries were constructed for IMDB, based on a recent keyword search benchmark [42]. We omitted 4 queries from [42], because they could not be translated to our query model. Our workload includes queries containing 2-11 patterns in total: 0-4 relation patterns, 1-7 string patterns, and 1-4 class patterns, see Table 2.

Note, since we extracted 1-grams only, every string pattern with a phrase of length n is decomposed into n string patterns. So, each string pattern captures exactly one word.

In our subsequent analysis, we rely on the number of patterns as an indicator for query complexity. We expect queries with a larger number of patterns to be more “difficult” in terms of both accuracy and efficiency. That is, accurate estimates may be harder to obtain and require additional computation.

However, most crucial are the dependencies between query patterns: we observed that there are more correlated patterns in IMDB, e.g., `info` (class `Movie`) and `title` (class `Movie`). Queries in DBLP, on the other hand, often include, e.g., `name` (class `Author`) and `label` (class `Title`) patterns, for which we could not measure any significant correlations.

Table 2 gives an overview of the query load, while example queries are given in Listing 6. All queries can be found in our appendix, see Section A.2.

Listing 6: Example queries for IMDB and DBLP benchmark. Variables are pink, keywords green, and classes as well as predicates are black.

```

1 // IMDB query // DBLP query
2 <x,type,Title> <x,label,"clustering">
3 <x,title,"star"> <x,label,"mining">
4 <x,title,"trek"> <x,year,"2005">
5 <x,cast_info,c> <x,type,Article>
6 <c,type,Cast_info> <x,author,y>
7 <c,role,r> <y,type,Person>
8 <r,name,rn> <y,name,"nikos">
9 <r,type,Char_name>
10 <c,person,p>
11 <p,type,Person>
12 <p,name,"brent">
13 <p,name,"spiner">

```

Systems. As string synopses we employed strategies proposed in [164]. That is, we obtained a random sample of 1-grams, top-k 1-grams, and stratified bloom filters (sbf) on 1-grams.

For selectivity estimating of the entire query, string patterns were integrated via: (1) *independence* (ind) or (2) *conditional independence* (bn) assumption. In the former case (independence assumption), selectivity of string and structured query patterns was estimated using string synopses and histograms. More precisely, the selectivity of structured query patterns was estimated similar to [87]. In the latter case (conditional independence assumption), selectivity estimation was performed using our BN^+ synopsis.

Combining string synopses with the independence/conditional independence assumption resulted in six different systems: $\text{ind}_{\text{sample}}$, $\text{ind}_{\text{top-k}}$, and ind_{sbf} rely on the independence assumption, and $\text{bn}_{\text{sample}}$, $\text{bn}_{\text{top-k}}$ as well as bn_{sbf} are BN^+ approaches.

Synopsis Size. We experimented with synopses of various sizes. The key factor driving the overall synopsis size was the employed string synopsis. The string synopsis determined the size of the (conditional) probability distribution for ind_* (bn_*), which was the most costly type of statistic. Other statistics, e.g., the BN^+ network structure, were negligible in terms of space.

We varied the number of 1-grams comprised by the top-k and sample synopsis, i.e., #1-grams per attribute $\in \{0.5\text{K}, 1\text{K}, 5\text{K}, 10\text{K}\}$. Regarding the sbf string synopsis, we captured up to $\{2.5\text{K}, 5\text{K}, 25\text{K}, 50\text{K}\}$ of the most frequent 1-grams for each attribute and varied the bloom filter sizes. This resulted in similar memory requirements for the sbf string synopsis. All systems loaded their synopsis into main memory.

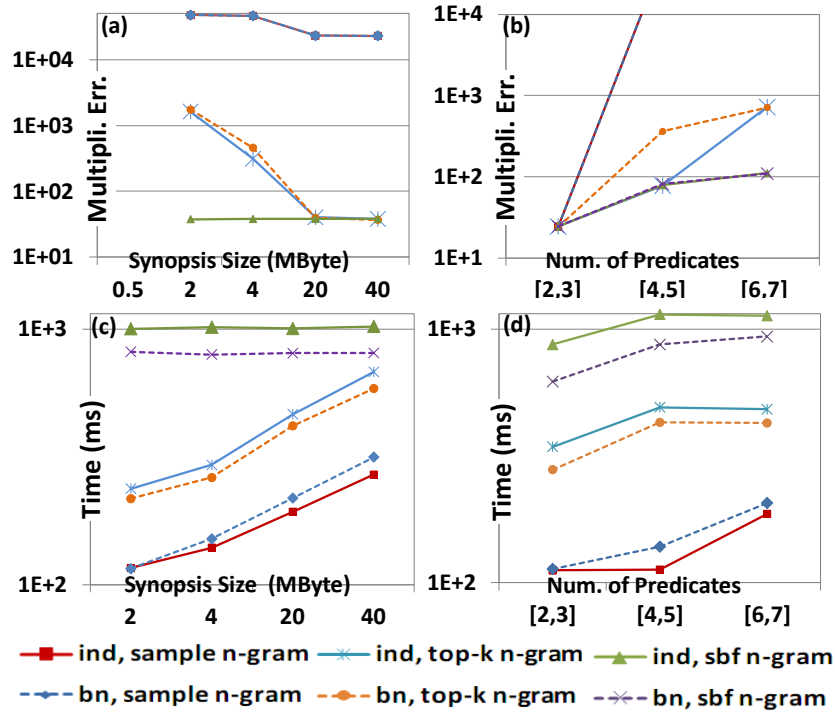
Overall, different string synopses (sizes) yielded different systems with $\{2, 4, 20, 40\}$ MByte of memory consumption, while no additional hard disk space was required. We observed that, while selectivity estimations become more accurate with greater size, no further improvements could be achieved, using synopses ≥ 20 MByte. In order to allow for the best accuracy and to illustrate this convergence, we report results from synopses with up to 40 MByte.

Implementation and Offline Learning. For bn_* systems, we used the BN^+ construction procedure, as presented in Algorithm 5. That is, we learned a model structure, capturing the most important correlations only. Then, we calculated model parameters (CPDs) based on sufficient statistics. String synopsis construction could be done efficiently: each synopsis, including sbf-based synopses, could be computed in less than one hour. Structure and parameter learning for bn_* combined took in the worst case up to three hours. Inferencing needed by our systems was done using a Junction tree algorithm [162].

As bn_* and ind_* systems rely on the same probability distributions for string patterns, parameters were shared. That is, for ind_* approaches we did not need a BN^+ model structure, but merely kept its marginalized parameters. Further, histograms for ind_* comprising relation and class statistics were constructed similar to [87]. Model structure (histograms) as well as parameters for bn_* (ind_*) were stored in a key-value store outside the database system – both were loaded into memory at start-up. Depending on the synopsis size loading took up to 3s.

We implemented all systems and algorithms using Java 6. Experiments were run on a Linux server with two Intel Xeon 5140 CPUs (each with 2 cores at 2.33GHz), 48GB RAM (with 16GB assigned to the JVM), and a RAID10 with IBM SAS 148GB 10K rpm disks. Before each query execution all operating system caches were cleared. The presented values are averages collected over five runs.

DBLP



IMDB

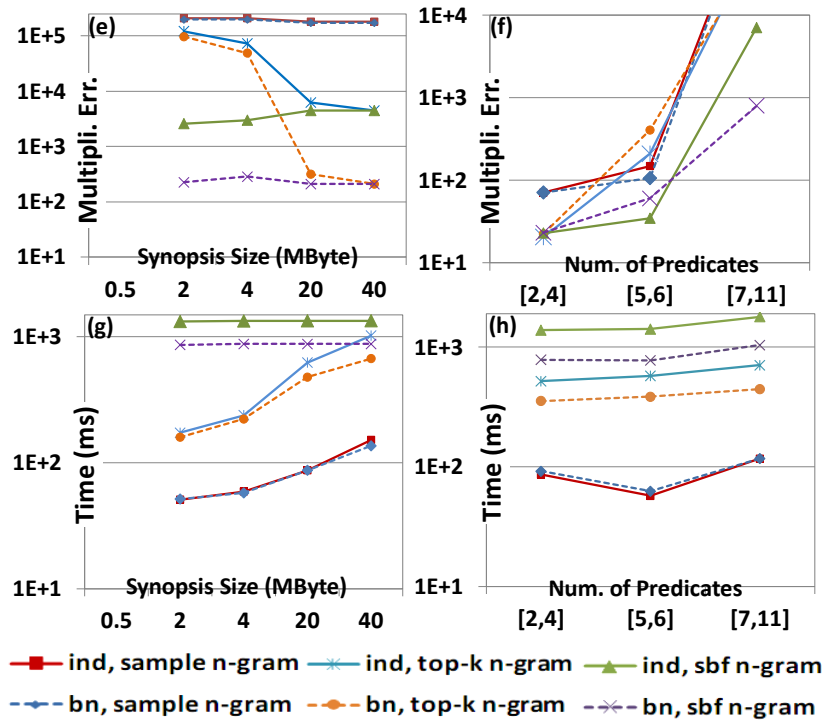


Figure 26: Evaluation results for DBLP and IMDB. All y-axes are in logarithmic scale.

4.4.2 Evaluation Results: Effectiveness

As metric for the selectivity estimation accuracy, we employed the *multiplicative error* (denoted as me), which was also used in previous work [48]. This error metric is defined as:

$$me(Q) := \frac{\max\{sel(Q), sel_a(Q)\}}{\min\{sel(Q), sel_a(Q)\}} \quad (17)$$

with $sel(Q)$ and $sel_a(Q)$ as exact and approximated selectivity for Q . Intuitively, the multiplicative error represents the factor at which $sel_a(Q)$ under- or overestimates $sel(Q)$.

Overall Results. Figure 26-a, -b (-e, -f) depict the multiplicative error for DBLP (IMDB). Best accuracy results were achieved by ind_* and bn_* having a size ≥ 20 MByte, because these synopses had sufficient memory space to capture most query keywords.

Further, the results confirmed our conjecture that the degree of data correlations has a significant impact on the accuracy differences between ind_* and bn_* approaches. That is, a high degree of correlation in the IMDB dataset translated to large accuracy differences. In contrast, the improvement that bn_* could achieve over the baseline was small for DBLP.

Last, comparing ind_* (bn_*) systems in terms of their string synopsis, we found that sampling-based approaches were outperformed by systems using top-k 1-gram synopses. Such systems, in turn, performed worse than sbf-based approaches. In fact, when using samples, the bn_{sample} system achieved results similar to those from ind_{sample} . This behavior is due to the fact that many keywords in query patterns were “missed” in the sample synopses. In these cases, both approaches rely on the same heuristic (leftbackoff strategy [164]) to calculate the probability for such keywords, which translates to large misestimates.

Synopsis Size. Figure 26-a and -e depict estimation errors vs. different synopsis sizes for DBLP and IMDB. Given a small synopsis (≤ 4 MByte), we observed that top-k and especially sample-based systems performed poorly, while accuracy for sbf-based approaches was fairly stable. With increasing synopsis size ($\in [4, 20]$ MByte), the performance of top-k 1-gram approaches converged to the accurate estimations, which were achieved by sbf-based systems. Differences in estimation quality can be explained by missed query keywords. More precisely, when missing a keyword, approaches have to rely on inaccurate heuristics for probability computation. The good and stable performance of sbf-based systems suggests that using stratified bloom filters is an effective strategy, which allows for an accurate estimation of most query 1-grams.

Data Correlations. Results obtained for IMDB and DBLP largely varied. For the IMDB dataset, bn_{sbf} could reduce errors of the ind_{sbf} approach by 93%, while improvements were much smaller given DBLP. For instance, for DBLP queries with string pattern name and label, there are no significant correlations in our

BN^+ . Thus, the probabilities obtained by bn_* were almost identical to the ones from ind_* . However, while ind_* led to fairly good estimates for the overall query load on DBLP, we could achieve more accurate selectivity computations via bn_* for specific “correlated” queries. For instance, for DBLP query Q1 we could approximate a 10% better selectivity estimation.

Query Size. Figure 26-b and -f show the multiplicative error for a varying number of query patterns. We noticed the error to increase in the number of patterns. This effect is expected, as more query patterns (hence more “difficult” queries) lead to an increasingly error-prone probability estimation.

An interesting observation is that ind_* outperformed bn_* for some queries – see IMDB queries with 5 patterns and DBLP queries with 4 patterns (Figure 26-b and -f). For instance, given IMDB query Q28, ind_{top-k} achieved 13% better results than bn_{top-k} . In such cases, string query patterns were translated to multiple values (1-grams) that are assigned to one single random variable. For processing these multiple assignments, bn_* employed value aggregation. However, the stochastic mode aggregation led to over-/underestimations for these queries due to inaccurate evidence weights. On the other hand, ind_* systems could approximate the probability simply via independence assumption.

Overall, we observed that while stochastic mode aggregation resulted in worse estimates for some queries, it led to better results on average.

4.4.3 Evaluation Results: Efficiency

During the second part of the experiments, we studied efficiency aspects of selectivity estimation with regard to varying synopsis sizes (Figure 26-c and -g) and query complexities (Figure 26-d and -h).

For all systems, our reported times represent solely the inference task (computation of Equation 16), while times for model construction and loading were omitted.

Overall Results. An important observation is that BN^+ inferencing did not have a decisive impact on the overall performance. Instead, the employed string synopsis was a key factor driving the efficiency: systems with sample-based synopses, bn_{sample} and ind_{sample} , were faster than approaches relying on top-k 1-gram synopses, which in turn outperformed sbf-based systems, bn_{sbf} and ind_{sbf} .

In fact, when employing the same string synopsis, bn_* approaches led to computation times comparable to those from ind_* . This can be explained with the lightweight model structure used by bn_* , which only captures the most important correlations. Further, our structure contained many tree-shaped parts, which could be processed efficiently through Junction tree inferencing.

Interestingly, we noticed ind_* systems to be even slower than bn_* in some cases. We explain this with: (1) the computational overhead of histogram-based estimation of structured query constraints for ind_* , and (2) with runtime advantages of bn_* due to stochastic aggregation. That is, fewer probability computations were performed by bn_* , because through value aggregation, the system could process several string patterns via one single inference task. On the other hand, ind_* ap-

proaches needed to compute the probability for each string pattern individually. For instance, bn_* needed 30% less computation time compared to ind_* for Q33 in the IMDB query load. This is because Q33 contains seven info string patterns that were aggregated by bn_* – leading to one random variable assignment.

String Synopses. Compared to other synopses, the time savings achieved by sample-based systems were possible due to missing 1-grams. However, such savings came at the expense of accuracy. If a particular query keyword is not included in a synopsis, heuristics are employed. In this case, the probability computation is done without the use of (conditional) probability distributions. Thus, no time-consuming marginalization was needed. Further, the missing 1-gram could not be added to the model “as evidence” for further inferencing.

Sbf-based systems performed worst. We explain this behavior with the computational overhead introduced by bloom filters. Further, as sbf synopses comprised a larger number of 1-grams, marginalization was more expensive. Note, with an increasing number of 1-grams to be managed, the performance of sample-based and top-k systems converged to the one exhibited by sbf-based approaches.

Synopsis Size. Figure 26-c and -g show selectivity estimation time vs. synopsis size. As expected, larger string synopses translated to bigger (conditional) probability distributions and hence, resulted in longer inference times.

Sbf-based approaches are an exception, as they provided a stable performance for different synopsis sizes. This constant estimation time was due to the fact that computational costs for sbf systems are largely determined by their bloom filters. In fact, we observed that costs only marginally depended on the overall number 1-grams.

Query Size. Figure 26-d and -h show that selectivity estimation times increase with query size. This is because each additional query pattern translated to more inferencing iterations and probability lookups that were needed by bn_* and ind_* systems.

4.5 RELATED WORK

For better effectiveness (accuracy), selectivity estimation approaches aim to avoid the uniform distribution assumption, the predicate value independence assumption, and the join predicate independence assumption (see Section 4.1.1).

One line of research employs *table-level* data synopses, i.e., data reduction techniques that capture joint distributions of attribute values within a table. Previous approaches utilize, e.g., histograms [48, 135] or wavelets [118]. Such table-level approaches are suitable for addressing the uniform distribution assumption and the predicate value independence assumption. However, the join independence assumptions can not be omitted, because table-level synopses are restricted to a single table and do not incorporate foreign-key relations.

Another line of research is concerned with *schema-level* synopses. Here, a synopsis does not only capture a single table, but also related tables, which are connected via foreign keys. Approaches based on graphical models [60, 162],

graph synopses [154], or join samples [10] have been proposed. Such solutions can avoid all three independence assumptions and thereby allow for effective selectivity estimates.

Our approach falls into this category. In fact, closest to our work are solutions based on PRMs [60, 162]. However, PRM-based approaches focus on relational data. We discussed in detail why PRMs are not directly applicable to schemaless Web data, see Section 4.3.1.2.

To summarize this discussion: A key problem is that PRM-based approaches assume queries with selection and join patterns, which are evaluated against explicitly specified tables. Queries in our setting, however, may not specify tables from which data shall be selected. In general, the effectiveness of PRM systems is greatly determined by the chosen data partitioning scheme. Addressing these shortcomings, we rely on a different template-based representation of BNs, which is well-suited for modeling probabilistic dependencies in Web data.

Further, no previous approach supports query patterns having large domains of textual values. In fact, some authors pointed out that the number of nominal values can be limited via clustering or, if possible, using feature hierarchies [60]. However, there is no work studying how clustering techniques may be integrated into a selectivity estimation framework, or how it may affect estimation effectiveness and efficiency. In this work, we build upon string synopses and show how they can be used in a template-based BN.

Another direction of related work is concerned with estimating the selectivity of string patterns [35, 98, 99, 107, 164]. Some approaches aim at substring and fuzzy string matching [35, 98, 107], while other systems target extraction operators, e.g., regular expression or dictionary-based operators [150, 164]. However, these works do not consider dependencies among multiple string patterns and/or structured patterns, which are evaluated against structured data. In this chapter, we showed that string synopses can be integrated into a template-based BN to deal with a conjunction of string/structured query patterns (hybrid queries).

In summary, our approach represents a novel schema-level synopsis, which is capable of handling hybrid queries over Web data.

4.6 SUMMARY

We targeted Research Question 2 in this chapter:

Research Question 2

How to allow for efficient and effective selectivity estimates on hybrid, schemaless Web data?

For this complex question, we validated two hypotheses: Hypothesis 3 and Hypothesis 4.

✓ Hypothesis 3

A template-based representation of BNs allows for effective and efficient selectivity estimation for *graph-structured* RDF data.

For the above hypothesis, we presented a novel template-based BN, coined BN^+ , which is well-suited for Web data. The BN^+ data synopsis resembles a schema-level data synopsis, which omits all three independence assumptions. At the same time, by means of our structure learning procedure as well as our fixed structure assumption, the BN^+ template model adheres to a lightweight network structure, which only captures key dependencies in the data.

✓ Hypothesis 4

String synopses can be integrated in template-based BNs and allow for effective and efficient selectivity estimation for *text-rich* RDF data.

Since Web data oftentimes contains text-rich attributes, the BN^+ synopsis must capture dependencies between/among those unstructured data elements and structured data elements. In order to efficiently capture such dependencies, we proposed the use of string synopses in the BN^+ synopsis – allowing to compress its sample spaces. This way, large sample spaces containing text values could be represented in a compact manner.

To validate effectiveness and efficiency of selectivity estimation based on our BN^+ data synopsis, we conducted experiments on real-world datasets. In fact, we could empirically show that, given there are dependencies between query patterns and text values, selectivity estimation effectiveness can be greatly improved. Moreover, we observed that this increased effectiveness does not come at the expense of efficiency, as the inferencing needed to consider the dependencies required only negligible overhead.

By means of our novel selectivity estimation, query optimizers can construct physical queries, which comprise rank-aware/approximate join operator over Web data – as proposed in this thesis. While we introduced a rank-aware join operator for distributed Web data in Chapter 3, we will present approximate join operators as well as a query processing pipeline for incremental query processing in the next chapter. Note that our approximate join operator not only requires selectivity estimation for integration in a physical query plan, but also for its internal join operation.

APPROXIMATE QUERY PROCESSING

 APPROXIMATE QUERY PROCESSING

Context of this Chapter. In this chapter, we introduce two approaches for *approximate query processing* over Web data. These approaches allow systems to *trade off result accuracy and result completeness for result computation time*. Moreover, our approaches are *complementary to each other*. Thus, a system may employ both works together. The following chapter is split into two main parts:

- ❶ On the one hand, we propose an approach for incremental and approximate query processing over Web data in Section 5.2. For this, we present a novel pipeline of operations, which *allows to process queries incrementally*. In particular, our pipeline consists of multiple join operators and data synopses, which are well-suited for schemaless Web data. This section is based on our previous publication [1].

In contrast to our LD-PBRJ operator that was discussed in Chapter 3, our incremental query processing pipeline does not consider result ranking. However, intermediate query results in our pipeline can be reported at any point in time. This way, large result sets can be processed efficiently, since systems have the possibility to answer queries quickly with initial results. In other words, systems can decide with what degree of accuracy to compute query results – as dictated by the given information need.

Additionally, query optimization techniques, which exploit our BN^+ selectivity estimation work (Chapter 4), can be used to optimize queries issued

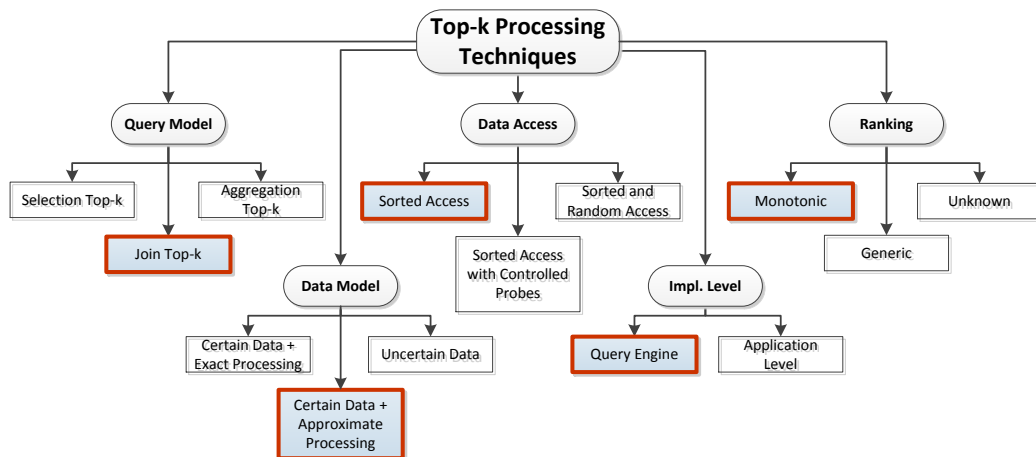


Figure 27: Classification of our *approximate top-k join processing* approach in Section 5.3.

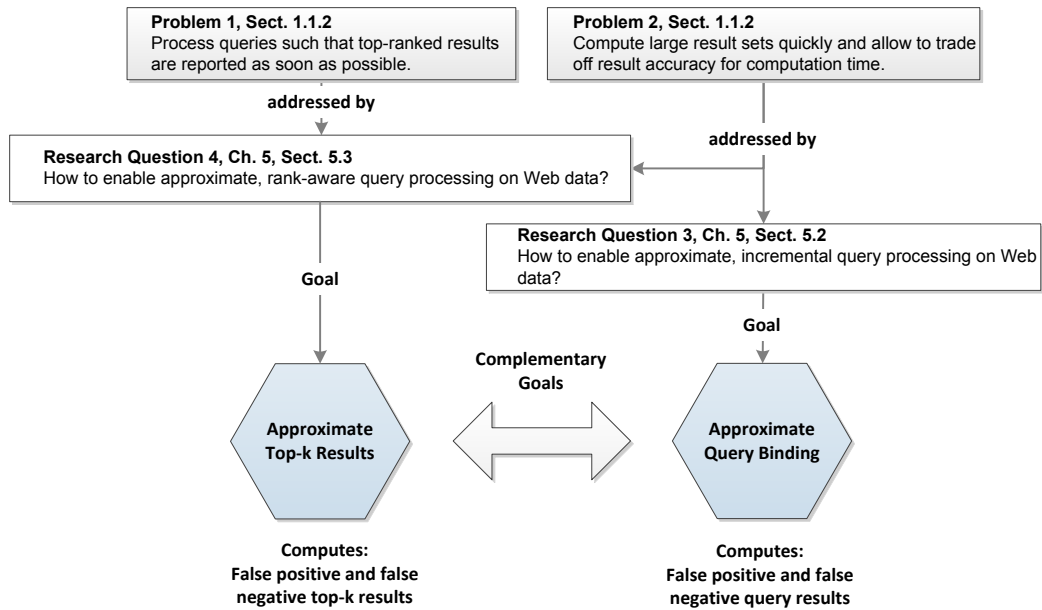


Figure 28: Overview: Approximate incremental approach in Section 5.2 versus approximate rank-aware approach in Section 5.3.

throughout the incremental query processing pipeline. In particular, we start our pipeline by computing entity queries, which oftentimes may resemble hybrid queries. Here, our BN^+ synopsis can provide effective selectivity estimates, thereby allowing the query optimizer to construct optimal physical query plans.

- ② On the other hand, we present an approximate top-k join processing approach in Section 5.3, which is based on our publication [4]. We highlight the classification of this rank-aware join operator in Figure 27. Recall, the dimensions for this classification have been introduced in Section 2.3.1.
 - As done for the LD-PBRJ operator, we target the join top-k problem. More precisely, we introduce a new approximate top-k operator, which can be integrated in physical query plans. In particular, we can employ our BN^+ selectivity estimation approach for the necessary cost estimates of the query optimizer.
 - Further, we require join inputs to be accessible via sorted accesses. In contrast to the LD-PBRJ operator, we do not need a specific sorted access implementation. In fact, given distributed Web data, we can employ our access plans presented in Chapter 3.
 - As before in Chapter 3, we need the ranking function to be monotonic.
 - Most importantly, contrary to the LD-PBRJ operator, our join operator approximates the top-k results. That is, we compute false positive and false negative top-k results, respectively.

Selectivity estimation, e.g., realized by the BN^+ selectivity estimation approach, is not only essential for the query optimizer, but also for the actual approximate top-k query processing. This is because, our approximate

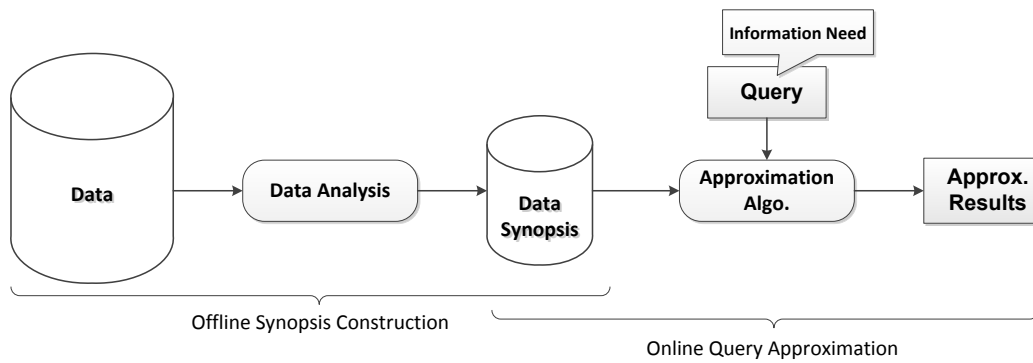


Figure 29: Generic approximate query processing approach [96, 112]. The data synopsis is constructed at indexing time and compactly summarizes the data. At query time, an approximation algorithm exploits the synopsis in order to compute approximate results for a given query.

top-k join operator exploits selectivity estimates, in order to judge if a partial binding will lead to a complete binding. Partial bindings, which have a low probability of leading to a complete binding, will be pruned during query processing.

Figure 28 illustrates the relation between our incremental query processing in Section 5.2 versus our approximate top-k query processing in Section 5.3. The first approach is *complementary* to the second approach. More specifically, our system in Section 5.2 computes false positive and false negative query results, respectively. That is, some query results may be *not valid*. In contrast, our approximate top-k query processing always computes *valid* query results. However, those results may be false positive or false negative top-k results. That is, some reported approximated top-k results may not belong to the exact top-k results.

Outline. We first give a brief motivation for approximate query processing in Section 5.1. The remainder of this chapter comprises two parts:

In Section 5.2, we discuss a system for incremental and approximate query processing. We outline research questions in Section 5.2.2. We propose our novel incremental query processing pipeline in Section 5.2.3. We discuss the evaluation results in Section 5.2.4. In Section 5.2.5, we outline the related work. Last, we summarize our findings in Section 5.2.6.

In Section 5.3, we introduce our new approach for approximate top-k join processing. More specifically, we discuss our research questions in Section 5.3.2. Our approximate top-k query processing approach is presented in Section 5.3.3. In Section 5.3.4, we discuss the evaluation. Further, we outline related works in Section 5.3.5. Finally, we conclude with Section 5.3.6.

5.1 MOTIVATION

Approximate query processing techniques constitute a popular class of processing techniques for large-scale data [58, 112]. Intuitively, approximate processing techniques ensure a quick response time – even for expensive queries – by ei-

ther omitting results or by reporting “roughly estimated” results. The first work addressing approximate query processing was published in [122].

The authors in [96] provide an overview of approximate query processing approaches – as illustrated in Figure 29. Intuitively, a *data analysis* algorithm is used to construct a *synopsis* at offline time from the data. At runtime, the data synopsis is employed to compute approximate results for a given query. Previous works exploited different kinds of synopses for approximate query processing. For instance, sampling, histogram, and wavelet-based synopses have been employed in various works [58]. The interested reader may find further details in [58, 96].

The motivation for applying approximate processing techniques is manifold and depends on the system, the users, and the information needs [58, 112]:

- ① Extensive datasets oftentimes reside on hard disks or tapes. Unfortunately, efficient data access on such media is still problematic. Approximation techniques feature compact data synopsis, which can provide the means to ensure fast response times.
- ② End-user applications frequently face information needs that do not require a high result accuracy. For example, given a Web search engine, users mostly visit only few top-ranked results – all remaining results are simply omitted. Thus, an engine may safely approximate low-ranked results, since only very few users will investigate them.
- ③ Data mining or decision support systems are very resource-intensive applications. At the same time, result accuracy or completeness is commonly not critical for such applications, because apply aggregation functions (e.g., sum, count, max, or min) are used to summarize results. Thus, approximation techniques can help to scale data mining and decision support systems to large datasets by roughly estimating the applied aggregation function.
- ④ Approximations may provide a query and dataset preview, respectively. That is, approximate query processing may be employed to allow users to gain a first insight into a dataset or issue an initial query.
- ⑤ Lastly, resources such as network bandwidth or storage space often make exact result computation impossible or very expensive. In contrast, approximate query processing allows to store the necessary data in a compact synopsis. In fact, this synopsis may be cached locally, thereby omitting network transfers.

Many of the above arguments are highly relevant for search over Web data. Most importantly, the amount of Web data is rapidly increasing – as motivated in the introduction in Chapter 1. So, many search systems will require slow data storage media such as hard disks. Moreover, end-users frequently search over Web data. Thus, systems have to deal with information needs, which can be addressed with via top-ranked results only. Lastly, Web data is commonly distributed over a space of data sources. Therefore, network communication costs, e.g., network latency, play a crucial role for Web search approaches.

In the following, we present two approaches for approximate query processing over Web data. While both can be seen as instantiations of the above framework, they differ in terms of their employed synopses as well as online query approximation.

5.2 APPROXIMATE INCREMENTAL QUERY PROCESSING

In the first section, we introduce an approach for incremental and approximate processing of hybrid queries over Web data. For this, we present a novel pipeline of operations, which allows to report results at any point in time to the users.

5.2.1 Introduction

Web Data Management. Efficient management of Web data bears novel challenges, which have attracted various research communities. In particular, several RDF stores have been implemented as *DB-based solutions* such as RDF-extensions for Oracle and DB2,³¹ Jena,³² Sesame,³³ or Virtuoso.³⁴

Further, *native solutions* for RDF like OWLIM,³⁵ HStar [39], AllegroGraph,³⁶ YARS [73], Hexastore [166], and RDF-3X [129] have been introduced.

Recently, also *IR technologies*, in particular inverted indexes, have been proposed for managing RDF data [173]. An overview over the various Web data management strategies can be found in [85, 139].

Problem. Unfortunately, all these systems focus on *computing complete and exact answers*. However, in a Web setting exact and complete query bindings (with billions of triples), lead to prohibitive response times – especially with respect to complex hybrid queries.

At the same time, as outlined above, many end-users have information needs, which can be addressed with incomplete and inaccurate results. In particular, the success of current Web search engines suggest that exact and complete results may not be needed. Recent studies estimate that 95% of all users investigate only the first 10 top-ranked results.³⁷

Thus, a more practical direction towards responsive and scalable solutions for Web-scale semantic data management is approximate query processing (together with sophisticated mechanisms for ranking, see Section 1.1.2). In this chapter, we focus on the problem of *approximate processing* and how to refine bindings *incrementally*. In other words, we give systems the freedom to decide *in which granularity to compute query results* – some information needs may require “more accurate” query results than others.

³¹<http://www-01.ibm.com/software/data/db2/>, retrieved 2014-02-10.

³²<http://jena.apache.org/>, retrieved 2014-02-10.

³³<http://www.openrdf.org/>, retrieved 2014-02-10.

³⁴<http://virtuoso.openlinksw.com/>, retrieved 2014-02-10.

³⁵<http://www.ontotext.com/owlim/>, retrieved 2014-02-10.

³⁶<http://franz.com/agraph/allegrograph/>, retrieved 2014-02-10.

³⁷<http://chitika.com/google-positioning-value/>, retrieved 2014-02-10.

5.2.2 Research Questions and Contributions

In the next paragraphs, we outline the research questions, hypotheses, and contributions we present throughout this section.

5.2.2.1 Research Questions and Hypotheses

In order to efficiently process hybrid queries over Web data, we want to exploit approximate and incremental query processing techniques. So, we ask the following research question:

Research Question 3

How to enable approximate and incremental query processing on schema-less Web data?

Notice, an overview of all research questions is given in Section 1.3. We address Research Question 3 by means of two hypotheses:

Hypothesis 5

Web data synopses and corresponding query processing algorithms allow for an *incremental* processing of hybrid queries.

Intuitively, above hypothesis expects Web data synopses to enable for an incremental processing of hybrid queries. In fact, users may stop the processing at any point in time and the system reports the currently known results. To target Hypothesis 5, we introduce a novel approach in Section 5.2.3 that features a pipeline of operators, which employ three data synopses: our neighborhood synopsis, our structure index synopsis, and our relation index synopsis. Moreover, we implemented this approach and conducted extensive experiments to empirically validate Hypothesis 5.

Hypothesis 6

Web data synopses and corresponding query processing algorithms enable an *approximate* processing of hybrid queries.

We predict with Hypothesis 6 that the same Web data synopses also allow for an approximate query processing. For this, we present a new approximate structure matching technique in Section 5.2.3. Furthermore, we introduce a novel approximate query processing strategy, which operates on a compact data synopsis. We empirically validate Hypothesis 6 by means of our experiments in Section 5.2.4.

5.2.2.2 Contributions

We propose a novel pipeline-based approach for processing hybrid queries over Web data. Our approach allows an “affordable” computation of an initial set of *approximate* results, which can be *incrementally* refined as needed. We thereby

allow a trade-off between result accuracy/completeness and query computation time. Our specific contributions are:

- *Contribution for Hypothesis 5*
Using a pipeline of four computation phases, query results can be incrementally processed – as required for the given information need. In the first phase, we compute potential entity matches. Then, structural relationships between those entities are checked in the subsequent phases. To the best of our knowledge, *this is the first work towards a pipelined processing of queries over Web data, which allows an incremental result computation.*
- *Contribution for Hypothesis 6*
For processing structured query patterns, we introduce a novel *approximate structure matching technique* based on our neighborhood join. In particular, we show how this approximate structure matching can be efficiently implemented via Bloom filters [27]. Another approximation is introduced for relation pattern matching: Instead of operating on the actual Web data graph, we approximate relation pattern matches over a compact data synopsis.
- *Contribution for Hypothesis 5 and Hypothesis 6*
We implemented our approach and conducted experiments by means of the LUBM benchmark [67] as well as DBLP data [13]. We show that our incremental approach preforms well with regard to time needed for computing exact and complete results. Furthermore, our approach also achieves promising results for approximate result computation. That is, a significant amount of processing time could be saved, while still producing query results with high precision and recall.

5.2.3 A Pipeline-based Approach for Approximate and Incremental Query Processing

5.2.3.1 Overview

Approach. For a hybrid query Q , we outlined the traditional query processing based on *graph pattern matching* in Section 2.2. As opposed to such an *exact* and *complete* query processing, an *approximate* procedure might output results, which only partially match the query Q . That is, a binding matches only some triple patterns comprised in Q . A query processing procedure is *incremental*, when results computed in the previous step are used for subsequent steps.

Figure 30 illustrates the main concepts and techniques of our approach. The Web data graph is captured by multiple data synopses: While attribute triples are stored in the entity index synopsis, relations triples are stored in the relation index synopsis. Moreover, the structure index synopsis [159] captures the overall structure in the Web data graph.

These synopses are employed in various operators in the pipeline, which we propose for query processing. We rely on sorted merge joins and reuse related query processing techniques [9, 166]. However, as opposed to exact and complete techniques, operations in our pipeline match the query against the data in an approximate way, thereby obtaining partially matching results (which are refined

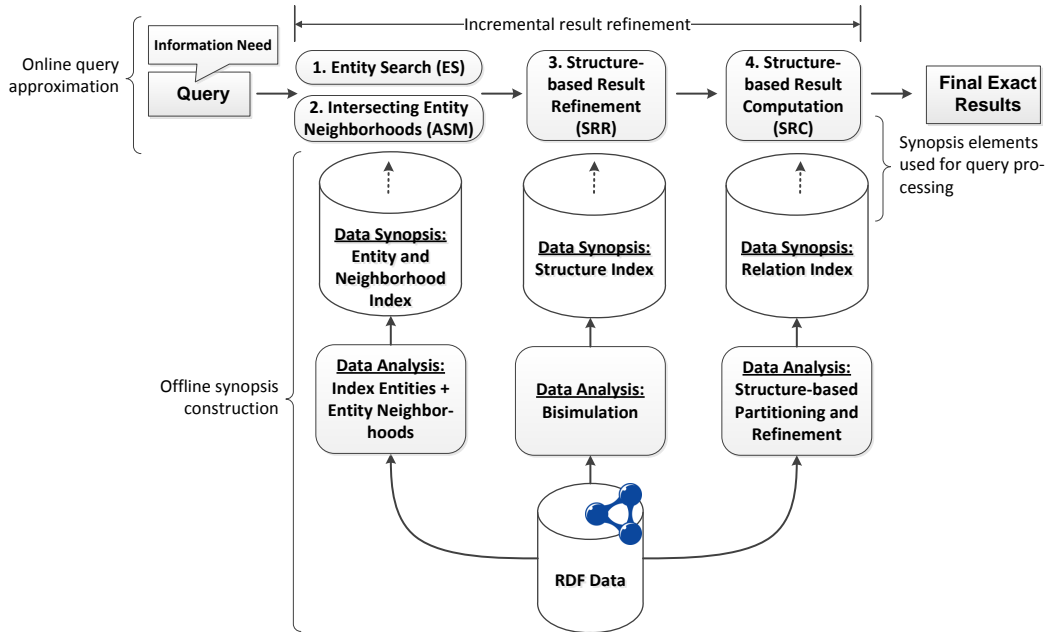


Figure 30: Offline data preprocessing and online query approximation, which is employed in our incremental query processing pipeline. Our approach can be seen as an instantiation of the general framework in Section 5.1. In particular, we instantiated the three framework complements: data analysis, data synopsis, and online approximation.

during the process). Note, instead of operating on all approximated results, it is possible to apply a cutoff or let the user choose the candidates at every step.

More precisely, our approach features steps as follows:

① *Entity Search (ES)*

We start by decomposing the query into entity queries and performing an *entity search*. During this search, we store results in sorted entity lists with a maximum length (cutoff). These results match only attribute query patterns and string query patterns.

② *Approximate Structure Matching (ASM)*

In the next step, we employ *approximate structure matching*. Here, we verify if the current results match the relation query patterns. For this, we compute the overlap of the neighborhood of the entities obtained from the previous step. That is, we verify if the entities are “somehow” connected – thereby we *approximately match the relation query patterns*.

③ *Structure-based Result Refinement (SRR)*

During structure-based result refinement, we further refine the previous results by searching for paths in the structure index, which *might* connect entities via relation query patterns.

④ *Structure-based Result Computation (SRC)*

Only in the final step (*structure-based result computation*), we actually use edges in the data graph to verify if these connections indeed exist, and

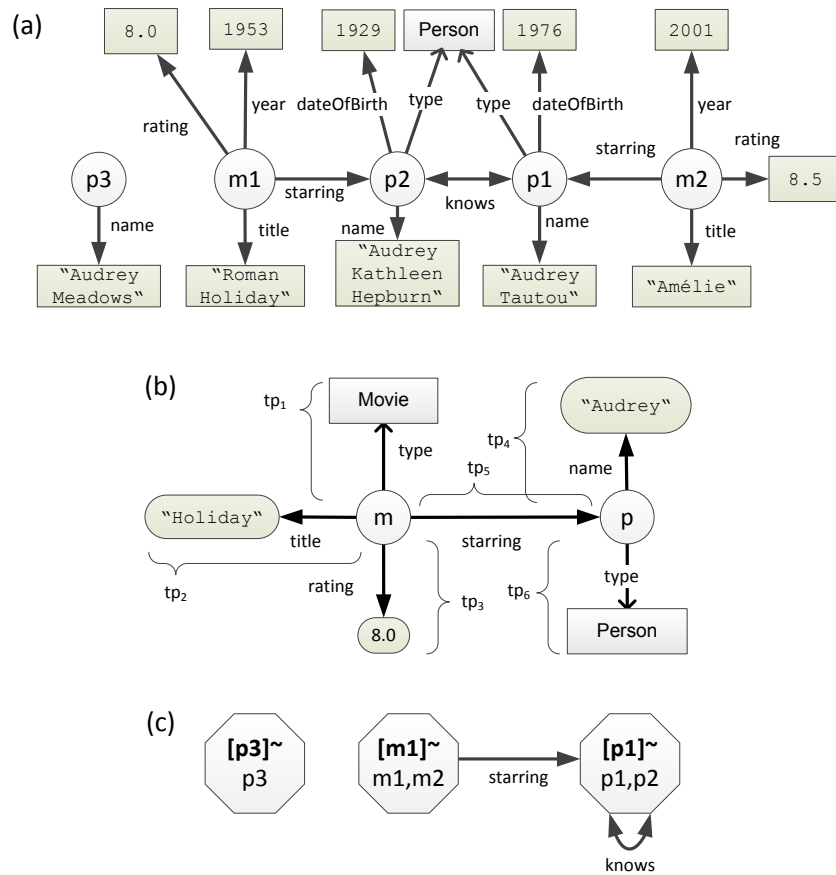


Figure 31: (a) A data graph, which is based on our running example in Figure 7. (b) Query graph based on Figure 8. Triple patterns tp_7 and tp_8 have been omitted for simplicity. (c) Structure index for the data graph in (a).

output the final results. Note, only these final results exactly match the query.

Example 43

Let us illustrate the above steps by our running example in Figure 31. Results of the refinement steps are also summarized in Table 3.

- We have two entity variables $\{m, p\}$ in our query, see Figure 31-b. During ES, we obtain initial bindings: variable m has potential match $\{m_1\}$ and variable p has potential matches $\{p_1, p_2, p_3\}$. Entities p_1 , p_2 , and p_3 are bindings for pattern tp_4 because of the contains semantic of string patterns (see Definition 9, p. 23).

- During the ASM step, we find that entity p_1 and entity p_2 are somehow connected with the other entities. This leads to two refined results: b_1 and b_2 , where

$$\begin{aligned}\mu_{b_1}(m) &= m_1, \mu_{b_1}(p) = p_1 \\ \mu_{b_2}(m) &= m_1, \mu_{b_2}(p) = p_2\end{aligned}$$

- In the SRR step, we check via the structure index if entity m_1 has starring as outgoing predicate. Further, we check if entities $\{p_1, p_2\}$ have starring as incoming predicate. Both conditions hold, so the result remains the same.
- Last, in the SRC step, we observe that the previous approximate techniques leads to one incorrect binding b :

$$\mu_b(p) = p_1 \text{ and } \mu_b(m) = m_1$$

Entity p_1 could not be pruned through ASM, because p_1 has a connection to m_2 via the starring relation.

Entity p_1 could also not be pruned via SRR, because when looking at the synopsis (structure index), p_1 exhibits the same structure as p_2 (i.e., p_1 and p_2 are both in the extension $[p_1]^\sim$ – see Figure 31-c).

Finally, by means of the SRC step, we find out that p_1 is actually not connected with m_1 via starring. So, entity p_1 can not contribute to a complete binding and the intermediate result b_1 is pruned.

Discussion. Our design is based on the observation that state-of-the-art techniques oftentimes only perform well with regard to queries with highly selective patterns (e.g., attribute patterns with a constant). However, query patterns containing many variables are more expensive. Considering Web-scale data graphs, these query patterns become prohibitive. That is, processing patterns such as $\langle x, \text{rdf:type}, y \rangle$ or $\langle x, \text{foaf:knows}, y \rangle$, requires millions of RDF triples to be retrieved. For dealing with Web-scale queries (having query patterns that might lead to large number of bindings), we propose a pipeline of operations. This pipeline starts with “cheap” query patterns to obtain an initial set of approximate results and incrementally refines the results via more expensive query patterns.

Work on data partitioning and indexing [9, 73, 166] is orthogonal and complements our solution. Moreover, we reuse existing techniques for exact and complete query processing based on sorted merge joins [9, 166]. Building upon these previous works, we present the first solution towards a *pipelined processing of complex queries* on Web data, which enables results to be computed approximately and incrementally.

In particular, our approach is the first *approximate technique* for querying RDF data, which is capable of trading result accuracy for computation time. That is, approximately matching results can be reported early and (if needed) result accuracy can be improved through several subsequent refinement steps. Compared

ES			ASM		
Binding	$\mu_b(m)$	$\mu_b(p)$	Binding	$\mu_b(m)$	$\mu_b(p)$
b_1	m_1	p_1	b_1	m_1	p_1
b_2	m_1	p_2	b_2	m_1	p_2
b_3	m_1	p_3			

SRR			SRC		
Binding	$\mu_b(m)$	$\mu_b(p)$	Binding	$\mu_b(m)$	$\mu_b(p)$
b_1	m_1	p_1	b_2	m_1	p_2
b_2	m_1	p_2			

Table 3: Approximated results after the ES, ASM, SRR, and SRC computation step. The corresponding query and data graph is shown in Figure 31.

to existing techniques, the structure refinement step (SRR) resembles a technique for approximate twig pattern matching [134]. The difference is that our structure index is a synopsis for general graph-structured Web data, while the synopsis employed in [134] is for hierarchical XML data only. Further, in contrast to previous techniques, we introduce an additional level of approximation: ASM. Our ASM phase exploits a novel approximate join operator that uses the notion of “neighborhood overlap” for structure matching.

As opposed top-k approaches, our *incremental approach* does not compute the best, but all approximate results, which are iteratively refined in several steps. In particular, we do not focus on ranking aspects and simply apply a predefined cutoff to prune large result sets.

5.2.3.2 Entity Search (ES)

Let us first describe offline entity indexing and afterwards online entity search.

Entity Indexing. Attributes $a \in \ell_a$ that refer to a particular entity are grouped together and represented as a document (ENT-doc) in an inverted index. We use structured document indexing – a feature supported in many IR engines such as Lucene³⁸ – to store entity attribute values in different fields:

- *Extension ID*
Field points to the entity’s extension.
- *Denotations*
Field contains the entity’s URI and name.
- *Attributes*
Field comprises a concatenation of attribute/value-pairs for that entity.

³⁸<http://lucene.apache.org/>, retrieved 2014-01-20.

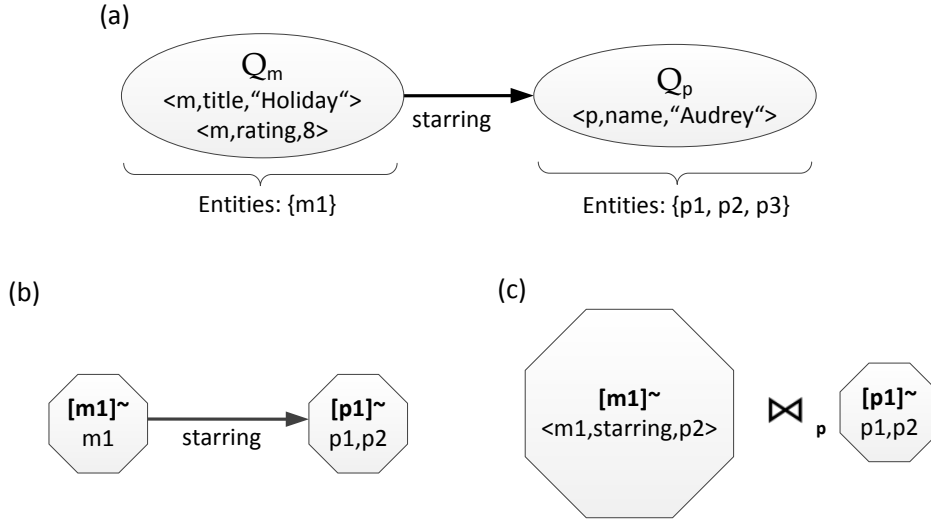


Figure 32: (a) The transformed query graph obtained in ES. (b) The structure index match computed in SRR. (c) Refinement in the SRC phase through joins along the structure index match.

- *k-neighborhood*
Field contains neighbor entities, which are reachable via paths with maximum length k .

Query Decomposition. We decompose the original query graph \mathcal{Q} into entity queries and a transformed query graph:

- *Entity Queries* \mathcal{Q}_E
We defined entity queries in Definition 20, p. 54. Intuitively, an entity query \mathcal{Q}_v constitutes a set of triple patterns, which share the common variable v at the subject position. In following, let \mathcal{Q}_E denote the set of all entity queries, which are comprised in a given query \mathcal{Q} .

Example 44

For instance, in Figure 32-a we have two entity queries: \mathcal{Q}_m and \mathcal{Q}_p . The former has m_1 as matching entity, while the latter has entities p_1 and p_2 as result.


- *Maximum Distance*
Let the distance between two queries, \mathcal{Q}_v and \mathcal{Q}_x , be defined as the length of the shortest path of relation patterns connecting the variables v and x . We denote this distance as $d_{\mathcal{Q}_v}(\mathcal{Q}_y)$.

Further, let $d_{\mathcal{Q}_v}^{\max}$ be the maximum distance between the entity query \mathcal{Q}_v in \mathcal{Q}_E and all other entity queries in \mathcal{Q}_E . More formally:

$$d_{\mathcal{Q}_v}^{\max} := \max \{d_{\mathcal{Q}_v}(\mathcal{Q}_y) \mid \mathcal{Q}_y \in \mathcal{Q}_E\} \quad (18)$$

- *Transformed Query*

The transformed query \mathcal{Q}' contains entity queries \mathcal{Q}_E as nodes and relation query patterns as edges. Query \mathcal{Q}' may be conceived as a compact representation of the original query \mathcal{Q} , where attribute/string query patterns in query \mathcal{Q} are “collapsed” into entity queries \mathcal{Q}_E . That is, each entity query node in \mathcal{Q}' represents a set of attribute/string query patterns from \mathcal{Q} .

 **Example 45**

The transformed query in Figure 32-a contains two entity queries, \mathcal{Q}_m and \mathcal{Q}_p , as nodes, which are connected via a starring edge. Further, entity query \mathcal{Q}_m contains two triple patterns:

$$\begin{aligned} \text{tp}_2 &= \langle m, \text{title}, \text{“Holiday”} \rangle \\ \text{tp}_3 &= \langle m, \text{rating}, 8 \rangle \end{aligned}$$

More specifically, a transformed query \mathcal{Q}' can be constructed as follows: We select an attribute/string query pattern $\langle v, a, o \rangle$ and create an entity query \mathcal{Q}_v for node v . Other attribute/string query patterns, which refer to the same variable v are added to \mathcal{Q}_v . Starting with this entity query, we construct the transformed query graph \mathcal{Q}' using breadth-first search (BFS) in \mathcal{Q} . We add visited relation query patterns as edges to the transformed query. If new attribute/string query patterns are encountered, we use them to create new entity queries.

During the traversal, the length of visited relation chains is recorded. This allows us to compute the distance for every entity query pair. That is, for every entity query \mathcal{Q}_x , we compute its distance $d_{\mathcal{Q}_x}(\mathcal{Q}_y)$ to another entity query \mathcal{Q}_y in \mathcal{Q}_E . Finally, the maximum distance $d_{\mathcal{Q}_x}^{\text{max}}$ is computed for every entity query \mathcal{Q}_x from this information.

Processing Entity Queries. Every entity query is evaluated by submitting its attribute/string query patterns as a query against the entity index. That is, query

$$\mathcal{Q}_v = \{\text{tp}_1 = \langle v, a_1, o_1 \rangle, \dots, \text{tp}_n = \langle v, a_n, o_n \rangle\}$$

is issued as a conjunction of its query patterns. Note, for our experiments, we used Lucene as the IR engine for indexing as well as computing entity query results. Given query \mathcal{Q}_v , we return a list of matching entities, where the maximum length of the list is less than a predefined cutoff value.

 **Example 46**

The query \mathcal{Q} in Figure 31-b is decomposed into the entity queries \mathcal{Q}_m and \mathcal{Q}_p , see Figure 32-a. Further, \mathcal{Q}_m and \mathcal{Q}_p are connected via relation starring, which leads to the transformed query \mathcal{Q}' in Figure 32-a.

To construct this transformed query, we start with pattern $\langle p, \text{name}, \text{“Audrey”} \rangle$ to create \mathcal{Q}_p . Then, we traverse the relation starring and meet a new entity variable m . Thus, we construct an entity query

$$\mathcal{Q}_m = \{\langle m, \text{title}, \text{“Holiday”} \rangle, \langle m, \text{rating}, 8 \rangle\}$$

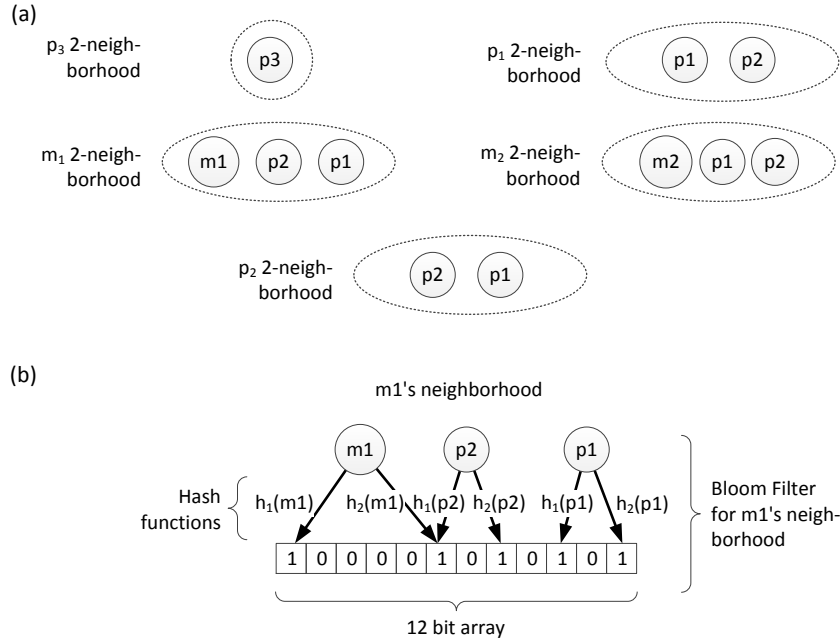


Figure 33: (a) Entity neighborhoods for data graph in Figure 31-a. (b) Bloom filter for m_1 's entity neighborhood. We employed two hash functions (h_1 and h_2) and a 12 bit array.

for variable m . Then, we add the relation starring as edge between Q_p and Q_m – leading to the transformed query Q' in Figure 32-a.

Last, for our entity search, we issue entity query Q_p and Q_m , which results in matching entities $\{p_1, p_2, p_3\}$ and m_1 , respectively.

5.2.3.3 Approximate Structure Matching (ASM)

So far, only entity query parts of query Q have been matched, while the relation patterns have not been processed. Typically, this structure matching is performed by joining the previously computed entities along their relation patterns. However, instead of such equi-joins (which would produce exact results), we propose to perform a *neighborhood join based on the intersection of entity neighborhoods*. In the following, let us introduce this novel concept for approximate structure matching and discuss suitable encoding and indexing techniques.

» Definition 25: Entity Neighborhood, Entity Neighborhood Join

Given a data graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \ell_a, \ell_r)$, the k -neighborhood of an entity $e \in \mathcal{V}_E$ is the set $E_{nb,k}^e \subseteq \mathcal{V}_E$, which comprises entities that can be reached from e via a path of relation edges with maximum length k .

The neighborhood overlap between entity e_1 and e_2 , denoted as $e_1 \cap^{nb} e_2$, is defined as intersection $E_{nb,k}^{e_1} \cap E_{nb,k}^{e_2}$.

The neighborhood join of the entity sets E_1 and E_2 , $E_1 \bowtie^{nb} E_2$, is an equi-join between all pairs (e_1, e_2) , with $e_1 \in E_1$, $e_2 \in E_2$. Further, two entities, e_1 and e_2 , are equivalent iff $e_1 \cap^{nb} e_2 \neq \emptyset$.

Managing neighborhood via Bloom filters. For every entity $e \in \mathcal{V}_E$, we compute its k -neighborhood via [BFS](#). Then, all elements in this neighborhood (including entity e) are stored in the entity index using the neighborhood-field (see the entity index description in Section [5.2.3.2](#)). At runtime, we represent entity neighborhoods as Bloom filters [\[27\]](#).

Bloom filters are space-efficient, probabilistic data structures that allow for testing whether an element is a member of a set. During this membership test false positives are possible. However, false negatives can not occur. The error probability for false positives is: $(1 - e^{-f \frac{n}{m}})^f$, where m is the size of the Bloom filter in bits, n is the number of elements in the set, and f is the number of hash functions used [\[27\]](#).

Example 47

The 2-neighborhoods for entities in our data graph (Figure [31-a](#)) are depicted in Figure [33-a](#). For instance, entity m_1 has the set

$$E_{nb,2}^{m_1} = \{m_1, p_2, p_1\}$$

as neighborhood. More precisely, m_1 's neighborhood is obtained by [BFS](#): first, we reach the 1-hop neighbors $\{p_2\}$, and then the 2-hop neighbors $\{p_1\}$. The bloom filter encoding of m_1 's neighborhood is given in Figure [33-b](#). For this, we employ two hash functions (h_1 and h_2) and a 12 bit array.

Approximate matching via Bloom filters. Checking for a connection between entity query Q_{e_1} and Q_{e_2} could be done by performing equi-joins over all possible relation paths (up to length k) between entity e_1 and e_2 . Unfortunately, this quickly becomes very expensive for large data graphs.

In contrast, we propose to check for these connections in an approximate fashion via a neighborhood join: $E_1 \bowtie_{E_{filter}}^{nb} E_2$. Note, this join solely operates on the Bloom filters associated with the entities. That is, our neighborhood join does not require retrieval and join of triples from the actual data graph. The join is evaluated by processing $e_1 \bowtie^{nb} e_2$ for all $e_1 \in E_1$ and $e_2 \in E_2$ in a nested loop manner – using the filters of elements in E_1 or E_2 , denoted by E_{filter} .

More precisely, for processing the join $e_1 \bowtie_{e_2}^{nb} e_2$, we evaluate if $e_1 \in E_{nb,k}^{e_2}$ by means of the bloom filter for entity e_2 . Performing neighborhood joins this way requires that the neighborhood of e_2 covers e_1 . In other words, the neighborhood index parameter k must be larger than or equal to the relation path length between e_2 and e_1 . For checking connections between entities in the sets E_1 and E_2 , along a chain of k relation patterns, only one set of Bloom filters has to be retrieved to perform exactly one neighborhood join. With the traditional approach, however, $k - 1$ binary equi-joins would have to be performed [\[85, 139\]](#).

The approximate structure matching (ASM) procedure based on our neighborhood join is illustrated in Algorithm [6](#). We search for a “center” entity query, Q_{center} , in Line [2](#). Q_{center} has the lowest eccentricity³⁹ and is used as starting point. From query Q_{center} , we process the neighbor entity queries in the trans-

³⁹The eccentricity of a vertex in a connected graph is given by its maximum distance to any other vertex in that graph [\[167\]](#).

Algorithmus 6 : Approximate structure matching (ASM) based on neighborhood join processing.

Input : Transformed query Q' . Let every entity query $Q_e \in Q_E$ be associated with a set of entities E_{Q_e} .

Output : Table A , where each row represents an approximated query result.

```

1 begin
  // initialize center and filter query
2   $Q_{center} \leftarrow \arg \min \{ \text{ECCENTRICITY}(Q_e) \mid Q_e \in Q_E \}$ 
3   $Q_{filter} \leftarrow Q_{center}$ 
4   $A \leftarrow E_{Q_{center}}$ 
5  while  $\exists Q_e \in Q_E : \neg \text{VISITED}(Q_e)$  do
6     $Q_{neighbor} \leftarrow Q_E$  found via DFS along a path of relation patterns
      starting at query  $Q_{center}$ 
      // set new query for the neighborhood join (if necessary)
7    if  $d_{Q_{filter}}(Q_{neighbor}) > k$  then
8       $Q_{filter} \leftarrow Q_{lastSeen}$ , where  $Q_{lastSeen}$  is the last seen query along
      the currently traversed DFS path
      // using the neighborhood join, we check
      // if entities are "somehow" connected
9       $A \leftarrow A \bowtie_{E_{Q_{filter}}}^{nb} E_{Q_{neighbor}}$ 
10 return  $A$ 

```

formed query Q' via **DFS** traversal, see Line 6. For every query $Q_{neighbor}$ in the current **DFS** path, we compute a neighborhood join between $Q_{neighbor}$'s entities and entities in the intermediate result A , see Line 9.

At the beginning, we marked the center node as Q_{filter} . Thereby we indicate that filters of $E_{Q_{center}}$ should be used for neighborhood joins "as long as possible", i.e., until $d_{Q_{filter}}(Q_{neighbor}) > k$, see Line 7. If bloom filters of $E_{Q_{center}}$ are not sufficient any more, we proceed with the filters of $E_{Q_{lastSeen}}$ – the entity query seen last on the current **DFS** path, see Line 8. This procedure aims at maximizing the reuse of bloom filters.

Example 48

We continue with our running example. Consider the query Q in Figure 31-b and the transformed query Q' in Figure 32-a. Entity neighborhoods for $k = 2$ are depicted in Figure 33.

Since both entity queries in Figure 32-a have an eccentricity of one, we could start with either entity query. For this example, we initialize $Q_{center} = Q_m$. Entities that match Q_m are added to A : $A = \{m_1\}$. By means of **DFS** from Q_{center} , we arrive at the 1-hop neighbor entity query, $Q_{neighbor} = Q_p$. Because it holds that

$$d_{Q_{center}}(Q_{neighbor}) = 1 < k = 2$$

we can use bloom filters from Q_{center} , i.e., Q_m , to compute the neighborhood join:

$$A \bowtie_{E_{Q_m}}^{nb} E_{Q_p}$$

Using this join, we can prune entity p_3 , because it is not connected to m_1 .

5.2.3.4 Structure-based Result Refinement (SRR) and Computation (SRC)

The ASM step leads to a set of approximated bindings. Every binding comprises entities that are *somehow* connected, i.e., connected via some “unknown” relations. During our SRR and SRC refinement, we want to find out whether they are actually connected via paths captured by our query patterns. More precisely, we propose the structure-based result refinement (SRR), which refines the previous results by using a structure index synopsis. Here, we check if bindings computed in the ASM step *could* match query relation patterns. Lastly, exact results are computed in the structure-based result computation (SRC) phase.

Structure Index for Graph-Structured Data. Structure indexes have been used for semi-structured and XML data [30, 38, 100]. A well-known concept is the dataguide [61], which is a structural description for rooted data graphs. Dataguide nodes are created for groups of data nodes that share the same incoming edge-labeled paths starting from the root. Similar to this concept, a structure index has been proposed for general data graphs [159]. Nodes in this index stand for groups of data graph nodes, which have the same “structural neighborhood”. Here, structural similarity is defined using the well-known notion of *bisimulation* [5]. Two nodes, v_1 and v_2 , are *bisimilar* (denoted as $v_1 \sim v_2$), if they cannot be distinguished by looking only at their outgoing or incoming “edge-labeled trees”. Pairwise bisimilar nodes form a node (so-called *extension*) in the structure index. These nodes are connected by relation edges from the data graph.

For our incremental query processing pipeline, we are only interested in grouping entities. Thus, for a given data graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \ell_a, \ell_r)$, we apply the bisimulation \sim to a subgraph $\mathcal{G}' = (\mathcal{V}_E, \mathcal{E}_R, \ell_r)$, which only comprises entity nodes and relation edges. So, one may conceive graph \mathcal{G}' as an “entity graph” that is based on data graph \mathcal{G} . A bisimulation on \mathcal{G}' leads to a set of extensions: $\{[v]^\sim \mid v \in \mathcal{V}_E\}$ with $[v]^\sim := \{v' \in \mathcal{V}_E \mid v \sim v'\}$. Extensions $[v]^\sim$ form a complete partitioning of entity nodes \mathcal{V}_E in entity graph \mathcal{G}' .

Given an entity graph \mathcal{G}' , we define the *structure index graph* \mathcal{G}^\sim as graph, where extensions are nodes and relations in \mathcal{G}' are edges. That is, an edge with label $r \in \ell_r$ connects extensions $[v_1]^\sim$ and $[v_2]^\sim$ iff \mathcal{G}' contains a triple $\langle v', r, v'' \rangle$, where $v' \in [v_1]^\sim$ and $v'' \in [v_2]^\sim$.

Example 49

The entities in Figure 31-a can be partitioned into three extensions – shown as nodes of the structure index graph in Figure 31-c.

For instance, entity p_1 and p_2 are grouped into the extension $[p_1]^\sim$ because they are bisimilar. That is, both have an incoming starring relation as well as an incoming/outgoing knows relation.

Algorithmus 7 : Structure-based result refinement (SRR) using the structure graph index based on [159].

Input : Transformed query Q' , entity queries Q_E , approximated query result from ASM phase in Table A, and structure index graph \mathcal{G}^\sim .

Buffer : EXT_{Q_e} is a two column table containing the entities for Q_E (computed during the ASM phase) and their extension.

Output : Approximated results in table A.

```

1 begin
2   foreach pattern  $\langle Q_{e_1}, r, Q_{e_2} \rangle \in Q'$  do
3     // get matching extensions for current query pattern
4      $E^\sim \leftarrow \{ \langle [v_1]^\sim, [v_2]^\sim \rangle \mid \langle [v_1]^\sim, r, [v_2]^\sim \rangle \text{ is triple in } \mathcal{G}^\sim \}$ 
5     foreach  $Q_e \in \{Q_{e_1}, Q_{e_2}\}$  do
6       if  $Q_E$  is entity query then
7         // check if ASM entities (for  $Q_E$ ) are associated with
8         // matching extensions
9          $E^\sim \leftarrow E^\sim \bowtie_{Q_e} \text{EXT}_{Q_e}$ 
10      if  $M = \emptyset$  then
11         $M \leftarrow E^\sim$ 
12      else
13         $M \leftarrow E^\sim \bowtie M$ 
14
15     // project refined entities on attributes in  $Q_E$ 
16      $A \leftarrow \pi_{Q_e \in Q_E}(M)$ 
17   return A

```

The structure index is well-suited for approximating query bindings with regard to relation patterns. This is due to the following lemma from [159]:

→ Lemma 1

If there is a query binding for a given data graph \mathcal{G} , there is also binding on the associated structure index graph \mathcal{G}^\sim . Moreover, matching extensions in the structure index graph comprise all matching entities in data graph \mathcal{G} .

A proof for the above lemma can be found in [159].

Structure-based Result Refinement (SRR). Given a query Q , Lemma 1 ensures that matching extensions in the structure index synopsis will contain all entity results in the data graph \mathcal{G} . Therefore, entities computed in the previous ASM step *can only contribute to valid query bindings, if they are comprised in matching extensions in the structure index graph*. Based on this observation, SRR performs two steps: (1) Compute matching extensions for the transformed query Q' in the structure index \mathcal{G}^\sim . (2) Check if the matching structure index extensions contain the previously computed entities.

Structure-based result refinement (SRR) is given in Algorithm 7. Matching extensions from the structure index \mathcal{G}^\sim are retrieved on Line 3 and buffered in E^\sim . Then, for each entity query Q_E , we check if entities for Q_E , which have been computed during the ASM step, are also contained in the matching extensions, see Line 6. For this, we use the extensions associated with these entities (as stored in ENT-doc, see Section 5.2.3.2) to construct the EXT_{Q_e} table and join this table with E^\sim . After processing all relation patterns, buffer M contains only entities, which are comprised in matching structure index extensions, see Lines 7-10. Finally, by projecting on the attributes in Q_E , we obtain the refined entities from M , see Line 11.

Example 50

The results computed during the ASM step are shown in Table 3. To refine these results, we start by searching the matching extensions in the structure index in Figure 31-c. For the relation pattern starring, we find the extensions $[p_1]^\sim$ and $[m_1]^\sim$. The former is a match for entity query Q_p , while the latter matches entity query Q_m . Entity query Q_p has two entities, p_1 and p_2 , which were computed by ASM. This leads to an EXT_{Q_p} table as follows:

EXT _{Q_p}	
Entity	Ext.
p_1	$[p_1]^\sim$
p_2	$[p_1]^\sim$

Since both entities are comprised in a matching extension, $[p_1]^\sim$, we can not prune any entity. For query Q_m , its ASM result m_1 is also associated with an matching extension, $[m_1]^\sim$. Thus, for our running example, SRR can't refine the ASM result and returns two approximated results:


$$b_1 = \left(\langle m_1, \text{type}, \text{Movie} \rangle, \langle m_1, \text{title}, \text{"Holiday"} \rangle, \langle m_1, \text{rating}, 8 \rangle, \right. \\ \left. \langle p_1, \text{name}, \text{"Audrey"} \rangle, \langle m_1, \text{starring}, p_1 \rangle, \langle p_1, \text{type}, \text{Person} \rangle \right)$$

$$b_2 = \left(\langle m_1, \text{type}, \text{Movie} \rangle, \langle m_1, \text{title}, \text{"Holiday"} \rangle, \langle m_1, \text{rating}, 8 \rangle, \right. \\ \left. \langle p_2, \text{name}, \text{"Audrey"} \rangle, \langle m_1, \text{starring}, p_2 \rangle, \langle p_2, \text{type}, \text{Person} \rangle \right)$$

Complete Structure-based Result Computation (SRC). In the last SRC phase, we compute exact query results. Note, only during this step, we actually perform joins on the data graph \mathcal{G} – all other phases employed data synopses.

To improve efficiency, we do not retrieve and join data along the query patterns in a standard way [85, 139]. Instead, we incrementally refine the results. That is, we reuse the structure index results and the entities associated with them – as stored in the intermediate result M (see Algorithm 7, Line 11). Intuitively, we

iterate over extensions in M and join their entities along the relation and class patterns present in the query, respectively.

 **Example 51**

Let us continue Example 50. We start with extension $[m_1]^\sim$ and its entity m_1 . As extension $[m_1]^\sim$ is connected with extension $[p_1]^\sim$ via the starring relation, we attempt to join m_1 with entities from $[p_1]^\sim$. This leads to one exact query result:

$$\mathbf{b} = \left(\langle m_1, \text{type}, \text{Movie} \rangle, \langle m_1, \text{title}, \text{"Holiday"} \rangle, \langle m_1, \text{rating}, 8 \rangle, \right. \\ \left. \langle p_2, \text{name}, \text{"Audrey"} \rangle, \langle m_1, \text{starring}, p_2 \rangle, \langle p_2, \text{type}, \text{Person} \rangle \right)$$

5.2.3.5 Theoretical Analysis

In the following paragraphs, we analyze our approach from a theoretical perspective. In particular, we will give a runtime complexity for each phase in Figure 30.

Consider a query Q , which has a query graph $\mathcal{G}_Q = (\mathcal{V}^Q, \mathcal{E}^Q)$, and a data graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \ell_a, \ell_r)$. Then, time complexity for computing query Q 's exact result set \mathbf{B} is bounded by:

$$O\left(\eta^{|\mathcal{E}^Q|}\right)$$

where η is the maximum number of matching triples for triple patterns in Q :

$$\eta := \max_{tp \in Q} |\{t \mid \mathbb{1}_M(tp, t) = \mathbf{T}, t \in \mathcal{E}\}|$$

Intuitively, this can be shown based on the following observation: Let \mathbf{B}_1 be a set of bindings for query Q_1 and let \mathbf{B}_2 be a set of bindings for query Q_2 . Then, the join $Q_1 \bowtie Q_2$ can be calculated in $O(|\mathbf{B}_1| \cdot |\mathbf{B}_2|)$ time. Note, by means of sorted indexes, fast merge joins achieve a near-linear behavior: $O(\eta \cdot |\mathcal{E}^Q|)$.

In contrast, our approach breaks query processing into four steps. With regard to our ES step, an entity query Q_E (comprised in query Q) might contain as many attribute/string query patterns as Q itself. That is, Q might be “one big” entity query. Further, these attribute/string patterns can be processed via fast merge joins over posting lists, see Section 5.2.3.2. Thus, ES has a time complexity of:

$$O(\eta \cdot |\mathcal{E}^Q|)$$

During the ASM step, $|Q_E| - 1$ neighborhood joins have to be performed. Since neighborhood joins are executed in nested loop manner, we require a complexity:

$$O\left(\zeta_{ES}^{|Q_E|-1}\right)$$

where ζ_{ES} stands for the largest entity query result set:

$$\zeta_{ES} := \max_{Q_e \in Q_E} \{|\mathbf{B}_{Q_e}|\}$$

	Data (#Triple)	Data (MB)	EntityIdx (MB)
DBLP	12,920,826	2,084	2210
LUBM₅	722,987	122	142
LUBM₁₀	1,272,609	215	253
LUBM₅₀	6,654,596	1,132	1391

	RelIdx (MB)	StrucIdx (KB)	Schema (KB)
DBLP	2,311	132	28
LUBM₅	112	100	24
LUBM₁₀	198	80	24
LUBM₅₀	1,037	82	24

Table 4: Statistics for the employed data graphs and their data synopses.

In the SRR step, we use nested loop joins on the structure index for all relation patterns in \mathcal{Q} . Thus, SRR has a complexity of

$$O(\eta^{\sim} \cdot |\mathcal{E}^{\mathcal{Q}}| \cdot |\zeta_{ASM}|)$$

where η^{\sim} is the maximum number of matching structure index edges for relation patterns in \mathcal{Q} and $|\zeta_{ASM}|$ is the maximum number of entity bindings (for queries $\mathcal{Q}_e \in \mathcal{Q}_E$), which were computed by the ASM phase.

Finally, we iterate over SRR results and join them along relation/class patterns. So, our SRC step requires a complexity of

$$O(|\zeta_{SRR}| \cdot |\mathcal{E}^{\mathcal{Q}}|)$$

where $|\zeta_{SRR}|$ refers to the maximum number of entity bindings, which were computed by the SRR step.

5.2.4 Evaluation

In the following, we present *empirical performance results and analyze the efficiency and result accuracy trade-off* to inspect the incremental and approximate features of our approach. We thereby aim to validate Hypothesis 5 and Hypothesis 6.

5.2.4.1 Evaluation Setting

Systems. We implemented our incremental approach (called INC) based on vertical partitioning and sextuple indexing [9, 166].

Unfortunately, there is no suitable baseline for the approximate and incremental features of our INC system. That is, ASM is based on the novel neighborhood join. Further, there is no alternative to SRR. As discussed in Section 5.2.3.1, SRR

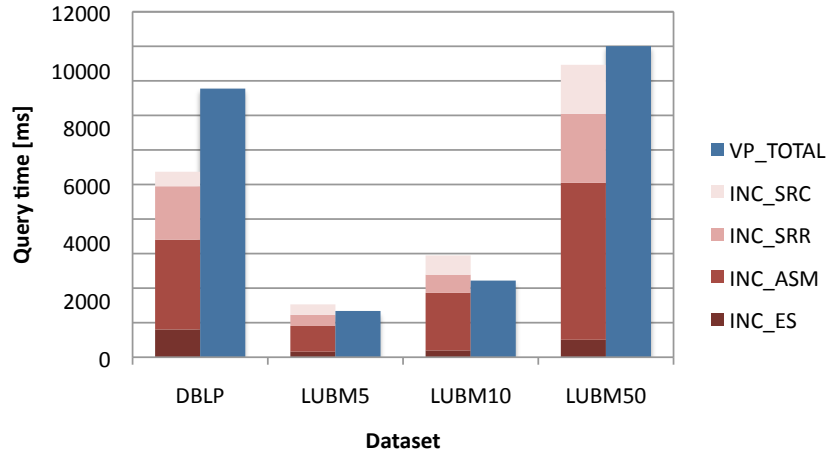


Figure 34: Query result computation times for different datasets.

is based on a summary, which is conceptually similar to synopses proposed for XML or relational data. However, it is not clear how to extend these concepts to graph-structured data and how to use them in a pipeline. *We therefore compare our approach with an exact and complete query processing system.* More precisely, we implemented an exact and complete sort merge join using the same data partitions and indexes (called VP).

Since optimization techniques, e.g., proposed in RDF-3X [129], are orthogonal to our work, all experiments were performed without such optimizations. That is, query processing was based on fixed query plans. For a given query, all approaches relied on the same plan.

All systems were implemented in Java 5. The bit-vector length and the number of hash functions, which we employed for ASM Bloom filters, were computed to reach an error probability of 0.1. Neighborhood indexes were created for $k = 3$.

For our experiments we used a Linux server with two Intel Xeon Dual Core 2.33GHz processors and 48GB of main memory (2GB were allocated to JVM). Data and indexes were stored on a Samsung SpinPoint S250 200GB SATA II disk. All reported values are averages over 10 runs. Before each query execution all operating system caches were cleared.

Data. We used the DBLP dataset, which captures real-world bibliographic information [13]. Further, we employed the LUBthM [67] benchmark to create three datasets for 5, 10, and 50 universities. An overview of our datasets and their corresponding data synopses is depicted in Table 4. Note, while our structure graph indexes are bigger than a typical data schema, they are much smaller than the underlying data graphs.

Queries. For studying the approaches in a principled way, we generated benchmark queries via random sampling. More precisely, we generated queries ranging from simple path-shaped queries to complex graph-shaped queries. For this, we used parameters as follows: the maximum number of constants con_{max} , the maximum number of paths $path_{max}$, the maximum path length l_{max} , and the maximum number of cycles cyc_{max} in the query graph. We sampled con-

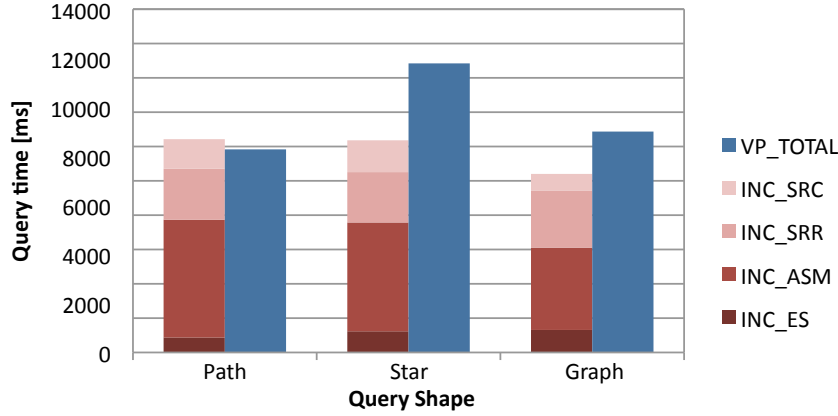


Figure 35: Query result computation times for different query shapes.

starts from attribute values in the data graph. Further, we sampled paths and cycles from data graph triples. The parameters used in the experiments are: $con_{max} = 20$, $path_{max} = 6$, $l_{max} = 3$, and $cyc_{max} = 2$. This resulted in a comprehensive query load of 80 queries.

Metrics. We measure system efficiency by means of *result computation time*. For measuring effectiveness of the INC approach, we compute result set *precision*:

$$precision := \frac{|\text{correct results} \cap \text{results retrieved}|}{|\text{results retrieved}|}$$

An intermediate query binding, which is computed during the ES, ASM, or SRR phase, is correct if it contributes to an exact result. For a query Q , the precision of an ES result set is defined as average precision over all result sets from entity queries in Q .

Notice, we do not report recall values, since the recall is always 1. *This is because all our pipeline steps solely check necessary conditions. So, no intermediate result that contributes to a complete result will be pruned.*

5.2.4.2 Evaluation Results

Efficiency: Overall Results. With regard to the INC approach, we decomposed the total result computation time into times for the ES, ASM, SRR, and SRC step. Further, we computed the average computation times over our query load – as illustrated in Figure 34 and Figure 35. We observed that the ES time is only a small fraction of the total computation time. Times for SRR and SRC, in contrast, were much larger. In fact, ASM constitutes the largest time share.

More specifically, the ES, ASM, and SRR step requires only 6%, 71%, and 84% percent of the total computation time, see Figure 34. These results suggest that users can obtain an approximated result set in a small fraction of time via ES, ASM, or SRR – depending on the desired result accuracy (discussed below). We explain these differences with our iterative processing of expensive query predicates. In particular, we noticed query processing for ES to be very fast, because the attribute/string patterns frequently led to few bindings and could be processed efficiently via merge joins.

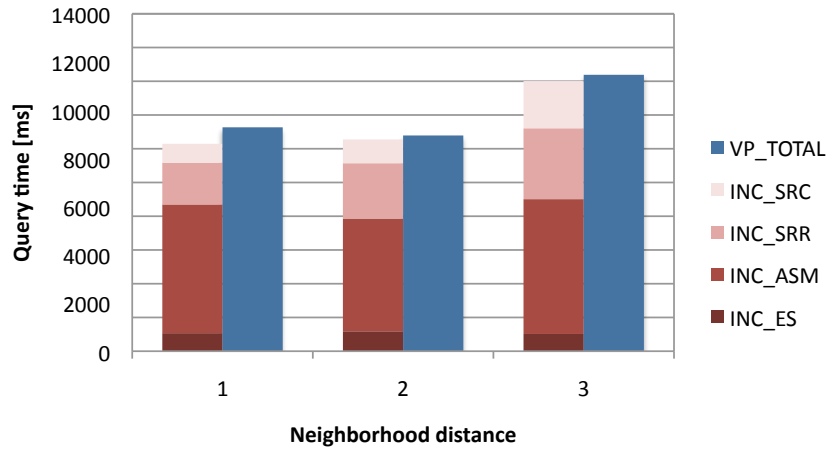


Figure 36: Effect of neighborhood distance k on result computation time.

Furthermore, when comparing total computation times, we observed INC to be slower than VP for LUBM5 and LUBM10, but faster for larger datasets such as LUBM50 and DBLP, see Figure 34. This is because large datasets lead to relation patterns being more expensive, which results in bigger gains of our approximate relation pattern processing. While these results might change with query optimization, they are still promising as they indicate that our incremental approach was able to effectively reuse intermediate results.

This confirms our Hypothesis 5: Our pipeline-based query processing approach allows for an effective incremental query processing over Web data.

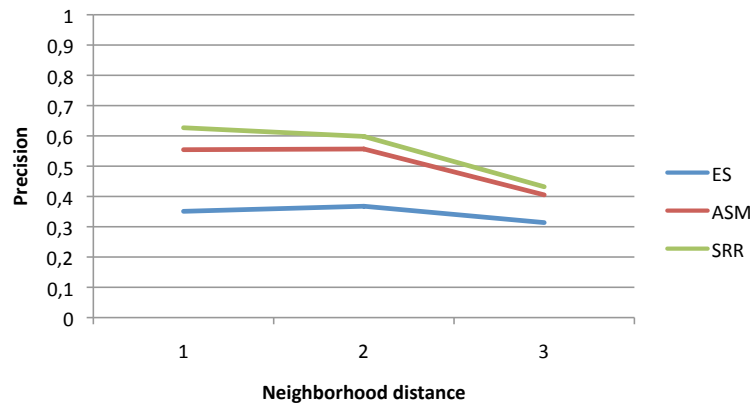


Figure 37: Effect of neighborhood distance k on result set precision.

Efficiency: Effect of Data Size. We measured the computation time versus different LUBM dataset sizes, see Table 4. As illustrated in Figure 34, the result computation time increased linearly with the size of the data – for VP as well as INC. Further, INC became more efficient with increasing data size. In particular, we observed the percentage of computation time needed for ASM to decrease with the data size. That is, the gain of ASM became “more clear” as the dataset grew larger. We argue this to be a key feature in the Web context: our ASM phase may help to quickly obtain initial result sets from a large data graph.

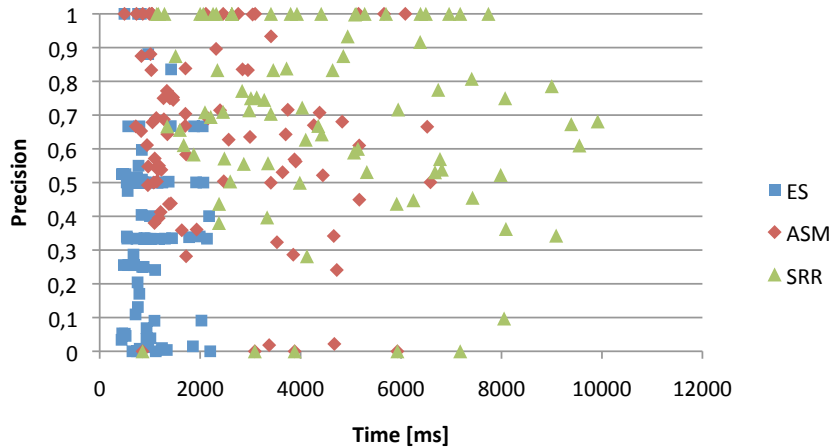


Figure 38: Result precision vs. result computation time.

Efficiency: Effect of Query Complexity. Considering query complexity, we classified our query load into three classes according to their shape: path-, star-, and general graph-shaped queries. As depicted in Figure 35, INC did not perform well on path queries. We noted ASM to be very expensive for this type of queries. This is because, the reusability of Bloom filters was low and new bloom filters had to be loaded frequently. Filter loading and nested loop joins became the bottleneck, resulting in slightly higher result computation times compared to VP.

Efficiency: Effect of Relation Pattern Path Length k . Furthermore, we categorized the queries according to the length of the longest relation pattern path, i.e., the “neighborhood distance” between entities. As shown in Figure 36, queries with longer relation paths required more time – for the VP as well as the INC approach. With regard to the INC system, we observed the time for the ASM phase to remain fairly constant. This suggests that the ASM step can be performed efficiently, even for long relation paths. Thus, ASM may help to deal with complex queries and provide initial results early.

Effectiveness: Effect of Relation Pattern Path Length k . Average precision for our different steps versus neighborhood distance k is illustrated in Figure 37. We noted the precision of ES to be relatively stable: 0.3 - 0.4. This can be expected, because k should have no effect on the entity search quality. For the ASM and the SRR phase, however, precision decreased in neighborhood distance k . We explain this effect with ASM/SRR approximations becoming more error-prone with increasing relation pattern path length. That is, approximations tend to be less accurate, the more relation patterns are considered.

Effectiveness: Time versus Precision. We show average computation time versus result set precision in Figure 38. Due to our incremental refinement, precision as well as result computation time increase with each additional step. Despite of some outliers, we observed a trend as follows: ES produced fast results at low precision – below 0.5 for most cases. Precision results could be largely improved through ASM. In fact, in 30% of the cases, ASM drove precision from 0.5 up to

0.8. For most of these cases (60%), the amount of additional result computation time was less than 10% of the total time.

Findings in the above paragraphs confirm the Hypothesis 6: The ASM step in our pipeline enables an effective approximate query processing over Web data.

5.2.5 Related Work

Much work in RDF data management targets orthogonal problems, namely data partitioning [9] and indexing [73, 166]. An overview over Web data management techniques is given in [85, 139]. We now discuss related approaches, which aim at approximate or incremental query processing:

Query Processing. Matching a query against a data graph is typically performed by retrieving triples and joining them along the query patterns – see also Section 2.2. Join processing can be greatly accelerated, when the retrieved triples are already sorted. Sorting is the main advantage of vertical partitioning [9] and sextuple indexing [166] approaches, which feature data partitioning and indexing strategies that allow fast (nearly linear) merge joins. Further efficiency gains can be achieved by finding an optimal query plan [129].

However, all these approaches aim at exact and complete query processing. In contrast, our pipeline-based approach allows a system to report approximated results – thereby saving a significant amount of processing time.

Approximate Query Processing. In the Semantic Web community, notions for *structural* and *semantic* approximation [44, 46, 47, 91] have been proposed, respectively. In these works, the focus lies on finding and ranking results that only approximately match a query. Moreover, strategies for *relaxing* a given query have been introduced [86, 88, 89, 90, 136]. Here, a query is modified, e.g., by removing one or more query patterns, in order to compute approximate query results.

In contrast, we propose a pipeline of operators, which *incrementally refines approximate results*. In particular, our approach allows to compute *exact* results – if necessary for a given information need. Moreover, throughout our pipeline, we one produce false positive results, but no false negative results. This is because all our processing steps are necessary, but not sufficient.

In DB community, approximate techniques have been proposed for “taming the terabytes” [18, 31, 58, 96]. Here, focus lies on *efficiency*. Instead of using the actual data, a query is processed over an appropriate synopsis (e.g., histograms, wavelets, or sample-based) – see also Figure 29. In fact, a suitable synopsis for XML data as been suggested [134], in order to compute approximate results for twig-pattern queries.

In contrast to approaches for flat relational data [58, 96], our SRR/SRC phase exploits a data synopsis, which is well-suited for graph-structured Web data [159]. Moreover, our novel neighborhood join allows an effective approximate structure matching based on a compact Bloom filter synopsis.

Incremental Query Processing. Related to our incremental approach is work on top-k query processing. Different algorithms for top-k query processing have been proposed [95]. Here, the goal is omit complete result materialization by stopping query processing as soon as the top-ranked results have been found.

In particular, we discussed an approach for Linked Data top-k processing in Chapter 3. Further, we will introduce an approximate top-k processing for Web data/queries in Section 5.3.

In contrast to these top-k query processing approaches, our incremental processing pipeline computes all results and is not restricted to the processing of top-ranked results. In fact, top-k query processing works are *complementary* to our approach.

5.2.6 Summary

We addressed the following research question in this section:

🔗 **Research Question 3**

How to enable approximate and incremental query processing on schema-less Web data?

For the above research question, we targeted two hypotheses by means of our novel pipeline-based query processing approach:

☑ **Hypothesis 5**

Web data synopses and corresponding query processing algorithms allow for an *incremental* processing of hybrid queries.

With regard to Hypothesis 5, we proposed a novel pipeline of operators, which allows to quickly compute an initial approximated query result. Then, these initial results can be refined using four processing steps. Intermediate results can be reported at any point in time during this refinement – as needed for the particular information need. We empirically validated our incremental query processing approach via extensive experiments on two RDF datasets.

☑ **Hypothesis 6**

Web data synopses and corresponding query processing algorithms enable an *approximate* processing of hybrid queries.

Concerning Hypothesis 6, we introduced two novel approximate query processing techniques in our query processing pipeline. On the one hand, we proposed the ASM phase, which allows to compute approximate relation pattern matches using our new neighborhood join. On the other hand, we proposed the SRR phase, which approximates query bindings by computing matches over the structure index synopsis. We empirically validated both phases in terms of their efficiency and effectiveness during our experiments. In particular, we observed

that initial results could be effectively refined via approximate structure matching with little computation time.

In the following section, we will introduce a complementary system that allows to approximately compute ranked query bindings.

5.3 RANK-AWARE APPROXIMATE QUERY PROCESSING

In the second part of this chapter, we present a novel approach for *approximate rank-aware query processing*. In contrast to our incremental query processing approach, we do not target query result approximation, but rather aim at an *approximation of the top-k results*. That is, all our query results are valid, but some may not belong to the exact top-k results.

5.3.1 Introduction

5.3.1.1 Motivation

As outlined in Chapter 1, queries over Web data frequently produce a large number of results. Reasons for this are twofold: (1) The amount of data on the Web drastically increased. This may strongly boost the number of query pattern matches. (2) There could be multiple possible data as well as query interpretations. Each data/query interpretation can be evaluated over the data graph – leading to an increase in the overall result set size for a given query. Therefore, the semantic search process incorporates result ranking as a key feature for an effective search over such large result sets [160, 7].

Ranking functions for queries over Web data oftentimes need to *incorporate query or user characteristics* [12, 36, 156]:

Example 52

Consider a user having the following information need: “Find movies with highest ratings, featuring an actress Audrey Hepburn, and playing close to Rome.” The corresponding query graph and ranking function is depicted in Figure 39.

Example 52 would require a ranking function to incorporate the movie rating, quality of keyword matches for “Audrey Hepburn”, and distance of the movie’s location to Rome. While one may assume that a higher movie rating is preferred by any user and query, *scores for keyword and location constraint dependent on query and user characteristics*.

For instance, in order to rank a binding for “Audrey Hepburn”, a function could measure the Levenshtein distance between that keyword and the binding’s attribute value, see Figure 39-c. Notice, given another keyword (e.g., only “Audrey”), the very same attribute value would yield a different score. Further, depending on the user’s geographic knowledge of Italy, she may have different notions of “closeness” to Rome, e.g., distance ≤ 100 km (see Figure 39-c).

We generalize from Example 52: Whenever a hybrid query contains a string pattern, a ranking function should measure the quality of the keyword match in the data graph. To allow for user/query-dependent, our definition of a ranking function (Definition 13, p. 32) quantifies the relevance of a binding b with regard to the query Q and the user who issued that query.

For efficiently processing queries with ranked results over Web data, two recent works employed top-k join processing techniques [2, 115]. In particular, we

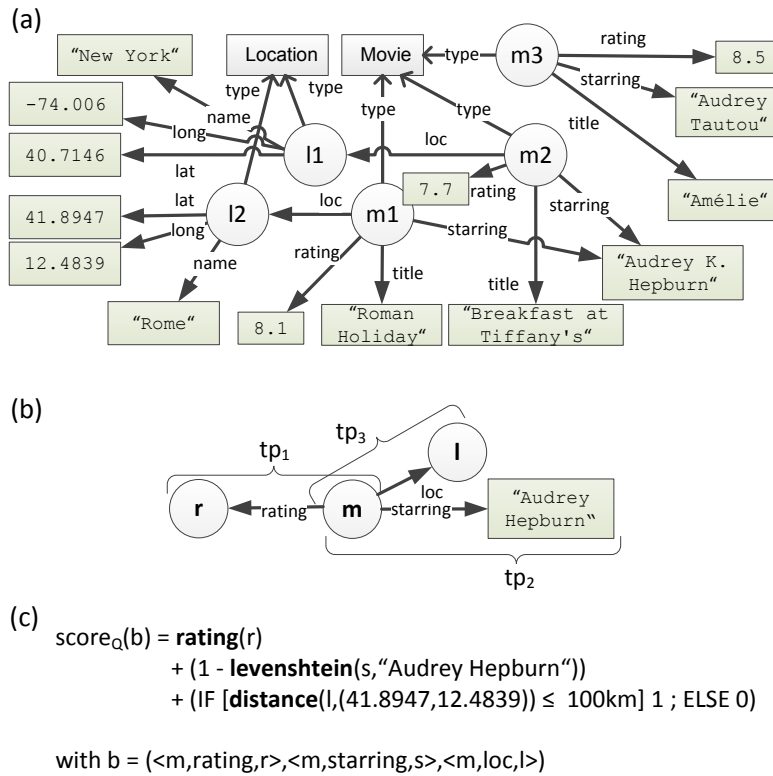


Figure 39: (a) A data graph based on our running example in Figure 7. (b) Query graph asking for a movie starring “Audrey Hepburn”, which is based on Figure 8. Note, several triple patterns have been removed for simplicity. (c) Scoring function, which aggregates scores for triple pattern bindings (bold): movie ratings, Levenshtein distance with regard to “Audrey Hepburn”, and distance of the movie’s location to Rome (lat: 41.8947, long: 12.4839) ≤ 100 km.

introduced our LD-PBRJ operator for top-k processing over distributed Web data in Chapter 3. Unfortunately, these approaches solely aim at computing exact and complete top-k results.

However, as outlined in Section 5.1, search on Web data is mostly performed by end-users, which *have information needs that do not require a high result accuracy or completeness*. In fact, result computation time is often the key factor.

Example 53

A system may compute top-50 movie results for Example 52. However, users oftentimes just visit one or two top-ranked results, until they discover a movie of interest. The remaining results are simply omitted.

Studies estimate that 95% of all Web search users visit less than the first ten top-ranked results.⁴⁰ Thus, users mostly pay attention to few top-ranked results, while low-ranked results are rarely investigated.

Above findings motivate the need for *approximated ranked results*. In simple terms, *a system should be able to approximate low-ranked (less important) results, in order to save computation time*.

⁴⁰<http://chitika.com/google-positioning-value/>, retrieved 2014-02-10.


5.3.1.2 Problem

Existing approaches for top-k processing over Web data compute *only exact and complete results* [2, 115], see Chapter 3.

Previous works for approximate top-k processing over relational databases [15, 16, 120, 151, 157] are not suitable for hybrid Web queries and Web data. This is because these works *assume complete ranking score statistics at indexing time*. Such statistics describe scores of partial and complete query bindings, respectively. So, in order to construct the necessary score statistics, one has to know all possible scores for partial/complete bindings of a given query at indexing time. Unfortunately, this leads to two major problems:

- *Problem 1: User-/Query-Dependent Ranking of Web Queries*

Query-/user-dependent ranking functions [12, 36, 156] are employed for many Web queries, e.g., keyword, spatial, or temporal queries [19, 40, 43, 111, 113]. Most notably, any hybrid query, which features a string pattern, requires a query-dependent ranking function. However, such *ranking scores are only known at runtime*.

 **Example 54**

Consider triple pattern tp_2 and tp_3 in Figure 39-b: Binding scores are decided by query characteristics (i.e., Levenshtein distance to query keyword “Audrey Hepburn”) or user characteristics (i.e., the user-defined distance to Rome). So, no score statistic can be computed for pattern tp_2 or tp_3 at indexing time.

- *Problem 2: Distributed and Frequently Changing Web Data*

Web data is commonly *highly distributed and frequently updated*. For instance, movie ratings for pattern tp_1 (see Figure 39-b) may be spread across multiple data sources – some of them could even be “hidden” behind SPARQL endpoints. Moreover, these sources may feature constantly updated rating scores. Thus, while constructing an offline statistic for rating scores is feasible, it comes with great costs in terms of maintenance. This problem is exacerbated by the fact that RDF allows for very *heterogeneous data*. For example, the rating predicate in query pattern tp_1 could be used to specify the rating of movies as well as products, restaurants etc. Thus, offline score statistics will grow quickly and become complex.

5.3.2 *Research Questions and Contributions*

In the next paragraphs, we discuss the research question, hypotheses, and contributions that we provide in this section.

5.3.2.1 *Research Questions and Hypotheses*

We aim at an approximate rank-aware query processing approach, which is well-suited for hybrid queries over Web data, by means of the following research question:

Research Question 4

How to enable approximate top-k query processing for hybrid queries over schemaless Web data?

We target Research Question 4 using two hypotheses:

Hypothesis 7

Given user/query-dependent ranking functions, we can learn score statistics for approximate top-k query processing by means of Bayesian statistics.

We expect Bayesian statistics to provide suitable means for learning score statistics during query processing. This way, we do not require information about ranking functions or score distributions at indexing time.

We validate Hypothesis 7 by describing our novel approximate top-k processing approach in Section 5.3.3. Moreover, we highlight the applicability of Bayesian statistics with regard to our approach in a theoretical analysis, see Section 5.3.3.6. In particular, we give bounds for the approximation error. Additionally, we empirically show that statistics learned via Bayesian statistics are suitable for approximate top-k processing using the evaluation in Section 5.3.4.

Hypothesis 8

Bayesian statistics allow for lightweight score statistics.

We expect Bayesian statistics to be well-suited for frequently changing Web data. More specifically, we expect Bayesian statistics to enable easily maintainable score statistics.

We provide a theoretical analysis of our approach to validate this claim. In particular, we show time and space complexity bounds in Theorem 5 and Theorem 6, respectively. Moreover, we empirically validate Hypothesis 8 by means of the evaluation in Section 5.3.4.

5.3.2.2 *Contributions*

In order to address the above research question and to target Hypothesis 7 and Hypothesis 8, we provide several contributions:

- *Contribution for Hypothesis 7*
This is the first work for approximate top-k join processing for hybrid queries over Web data. In particular, our approach learns score distributions using Bayesian statistics in a pay-as-you-go manner at runtime. This way, we allow user/query-dependent ranking functions to be employed – a key requirement for effective search via hybrid queries.
- *Contribution for Hypothesis 7 and Hypothesis 8*
We provide a theoretical analysis of our approach and formally show its efficiency and effectiveness. In particular, we show the distribution learning to have a constant space complexity and a runtime complexity bounded by the result size. Moreover, we give bounds for the approximation error of our approach.
- *Contribution for Hypothesis 7 and Hypothesis 8*
We implemented our approach and conducted experiments on two SPARQL benchmarks. Evaluation results are promising, as we could achieve time savings of up to 65%, while still having a high precision/recall.

5.3.3 Pay-as-you-go Approximate Top-k Join Processing

We now present an approximate top-k join approach, the so-called A-PBRJ, that is tailored towards the frequently changing Web data as well as hybrid queries and their user/query-dependent ranking functions.

5.3.3.1 Prerequisites

Similar to the LD-PBRJ operator in Chapter 3, we also require sorted accesses for our A-PBRJ approach. Moreover, we rely on Bayesian statistics for learning the score statistics at runtime.

Sorted Access. As presented in Definition 14, given a pattern tp_i , a sorted access sa_i retrieves matching triples in descending score order. Previous works on join top-k processing over Web data introduced efficient sorted access implementations for RDF stores [2, 115].

In this work, we do not require a specific sorted access implementation. In fact, given distributed Web data, we could exploit the sorted access implementation for the LD-PBRJ in Section 3.3.1

Example 55

Let us continue Example 52. Here, sorted accesses for the data/query graph in Figure 39 could be implemented in different ways. We also illustrate these sorted accesses in Figure 40-a.

- Given the keyword pattern $tp_2 = \langle m, \text{starring}, \text{“Audrey Hepburn”} \rangle$, a sorted access must materialize all triples, which have a value that contains “Audrey” or “Hepburn”. After materialization these triples

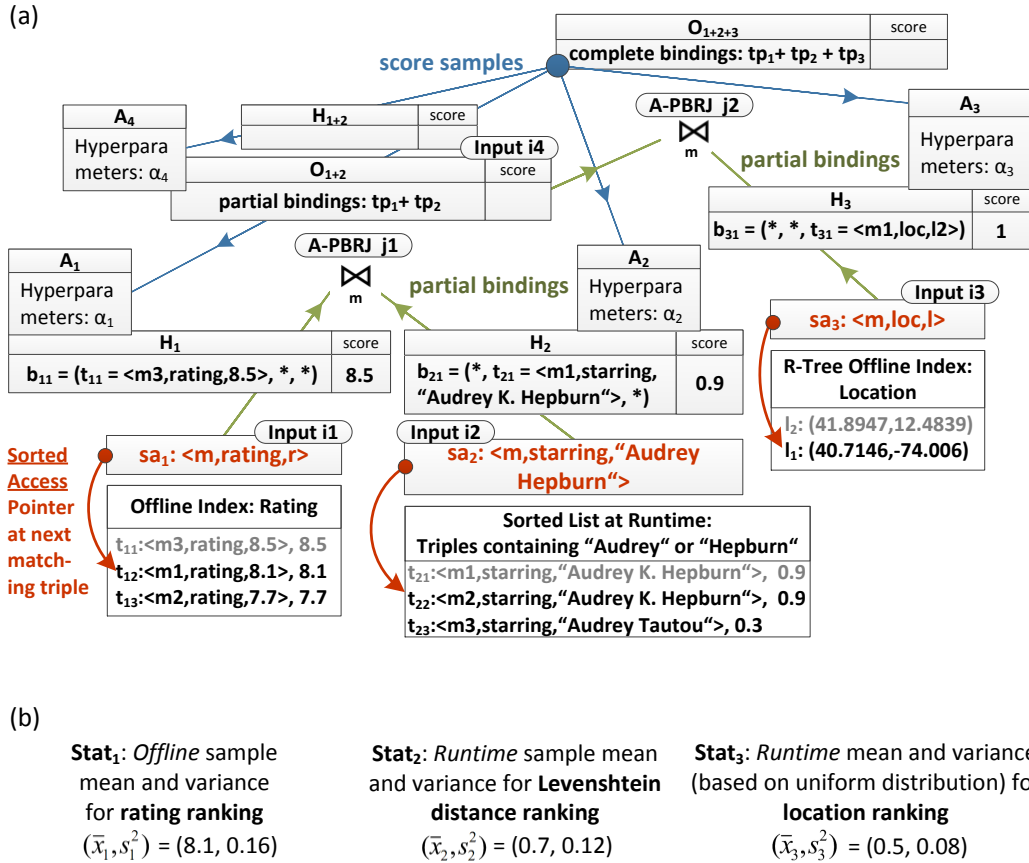


Figure 40: (a) Approximate rank join tree with three sorted accesses – one for each triple pattern in Figure 39-b. Two information flows occur in the tree: partial bindings (green) and score samples (blue). (b) Sufficient statistics calculated based on scores observed at indexing time ($stat_1$) and runtime ($stat_2$ and $stat_3$).

are sorted with descending similarity with regard to that keyword (e.g., measured via Levenshtein distance [26]).^a Thus, sorted access sa_2 loads three triples comprising “Audrey” or “Hepburn” and sorts them according to their Levenshtein distance to “Audrey Hepburn”.

- In contrast, for pattern $\langle m, loc, l \rangle$ an R-tree [69] on the attribute pair (lat, long) may be used. This offline computed index yields two location hits: l_1 and l_2 . While l_2 is an exact match (thus, triple t_{31} has max. score 1), l_1 is more distant from Rome. Note, location l_1 (triple t_{32}) is only loaded if needed, i.e., if join-2 pulls on sorted access sa_3 a second time.
- Last, an index for attribute *rating* can be constructed at offline time. For this, triples are stored with descending rating value. Then, the sorted access sa_1 can iterate over this list.

^aThe distance between two strings may be measured by various metrics. A commonly employed heuristic is the Levenshtein distance [26].

Partial bindings retrieved from sorted accesses are combined via joins. That is, an equi-join combines two (or more) inputs. This way, multiple joins form a tree. For instance, three sorted accesses are combined via two joins in Figure 40-a.

Score Statistics. We do not require offline knowledge about ranking functions or score distributions. Every score statistic needed is learned at runtime by means of the *pay-as-you-go* paradigm. More precisely, we exploit *conjugate priors* for learning the necessary probability distributions.

Let Θ be a set of parameters. One can model *prior beliefs* about these parameters in the form of probabilities: $\Theta \sim P(\Theta | \alpha)$ [22, 83]. Here, α is a vector of *hyperparameters* allowing to parametrize the prior distribution. Suppose we observe relevant data $\mathbf{x} = \{x_1, \dots, x_n\}$ with regard to Θ , where each $x_i \sim P(x_i | \Theta)$. Then, the dependency between observations \mathbf{x} and prior parameters Θ can be written as $P(\mathbf{x} | \Theta)$. We can estimate a *posterior* probability using the Bayes theorem, which captures parameters Θ conditioned on observed events \mathbf{x} . In simple terms, a *posterior distribution models how likely parameters Θ are, in light of the seen data \mathbf{x} and the prior beliefs* [22, 83]:

$$P(\Theta | \mathbf{x}, \alpha) \propto P(\mathbf{x} | \Theta) \cdot P(\Theta | \alpha) = \frac{P(\mathbf{x} | \Theta) \cdot P(\Theta | \alpha)}{\sum_{\Theta} P(\mathbf{x} | \Theta)P(\Theta)} \quad (19)$$

The term $P(\mathbf{x} | \Theta)$ is oftentimes conceived as a function of Θ , given fixed observations \mathbf{x} . Then, $P(\mathbf{x} | \Theta)$ can be written as *likelihood* function:

$$l(\Theta | \mathbf{x}) := P(\mathbf{x} | \Theta)$$

Following this interpretation, one may read Equation 19 as [22, 83]:

$$\text{posterior} \propto \text{prior} \times \text{likelihood}$$

Example 56

For pattern tp_1 in Figure 40-a, scores are based on rating values. So, we can compute sufficient statistics, i.e., a mean $\bar{x}_1 = 8.1$ and a variance $s_1^2 = 0.16$, for these scores at offline time, see $stat_1$ in Figure 40-b.

These statistics represent prior beliefs about the “true” distribution that is capturing only those scores for bindings of tp_1 that are part of a complete binding. In fact, only triple t_{12} and t_{13} contribute to complete bindings. Thus, only their scores should be modeled via a distribution. We update the prior beliefs using scores samples \mathbf{x} observed during query processing, thereby learning the true (posterior) distribution as we go.

Since we are interested in *unobserved* events x^* , we need the *posterior predictive distribution*, i.e., the distribution of new events given observed data \mathbf{x} [22, 83]:

$$P(x^* | \mathbf{x}, \alpha) = \sum_{\Theta} P(x^* | \Theta)P(\Theta | \mathbf{x}, \alpha) \quad (20)$$

An important kind of Bayesian priors are the *conjugate priors* [22, 83].

» **Definition 26: Conjugate Prior**

A prior distribution family \mathcal{D} for a parameter set Θ is called *conjugate* iff $P(\Theta) \in \mathcal{D} \Rightarrow P(\Theta | \mathbf{x}) \in \mathcal{D}$.

Intuitively, conjugate priors require the posterior and prior distribution to belong to the same distribution family. In other words, these priors provide a “computational convenience”, because they give a closed-form of the posterior distribution [22, 83]. Thus, posterior computation is easy and efficient for conjugate priors – enabling an efficient distribution learning at runtime.

5.3.3.2 Approximate Rank Join Framework

We extend the Pull Bound Rank Join (PBRJ) framework (shown in Algorithm 1), and propose a new approximate Pull/Bound Rank Join (A-PBRJ) framework that comprises three parts: a pulling strategy \mathcal{PS} , a bounding strategy \mathcal{BS} , and a probabilistic component \mathcal{PC} .

\mathcal{PS} determines the next join input to pull from [143]. The bounding strategy \mathcal{BS} gives an upper bound, β , for the maximal possible score of unseen join results [143]. Last, we use our new probabilistic component \mathcal{PC} to estimate a probability for a partial binding to contribute to the final top-k result.

Overview. The A-PBRJ is depicted in Algorithm 8. Based on Algorithm 1, we first check if k top-ranked complete bindings have been found (see Algorithm 8, Line 4). If so, the A-PBRJ terminates and reports the top-ranked results in \mathbf{O} . Otherwise, we attempt to produce further join results. That is, \mathcal{PS} selects an input i to pull from (see Algorithm 8, Line 5) and materializes a new partial binding b . Afterwards, we update the β bound via bounding strategy \mathcal{BS} .

 **Example 57**

In Figure 40-a, join j_2 decides (via strategy \mathcal{PS}) to first pull on sa_3 and load partial binding t_{31} . Then, join j_2 pulls on input i_4 (join j_1), which in turn pulls on its input i_1 (sa_1) loading binding t_{11} and afterwards on input i_2 (sa_2) loading t_{21} . The join attempt $t_{11} \bowtie t_{21}$ in j_1 fails, because $m_3 \neq m_1$.

In Line 8, \mathcal{PC} estimates the probability for a partial binding b leading to a complete top-k binding: *the top-k test*. If b fails this test, b will be *pruned*. That is, we do not attempt to join it and do not insert it in \mathbf{H}_i . Here, \mathbf{H}_i is a buffer that holds “seen” bindings from input i .

Otherwise, if the top-k test holds, b is further processed (see Lines 9-15). That is, we join b with seen bindings from the other input j and add results to output buffer \mathbf{O} . Further, b is inserted into buffer \mathbf{H}_i , see Line 10.

For learning the necessary probability distributions, \mathcal{PC} trains on seen bindings/scores in \mathbf{O} , see Line 12. Notice, we continuously train \mathcal{PC} throughout the query processing – every time “enough” new bindings are in \mathbf{O} , see Line 11. \mathcal{PC} requires parameter ω for its pruning decision. ω holds the the smallest currently known top-k score (see Line 15). ω is initialized as $-\infty$ on Line 2.

Algorithm 8 : Approximate Pull/Bound Rank Join (A-PBRJ).

Param. : Pulling strategy \mathcal{PS} , bounding strategy \mathcal{BS} , probabilistic component \mathcal{PC} .

Index : Sorted access sa_i and sa_j for input i and j , respectively.

Buffer : Output buffer \mathbf{O} . \mathbf{H}_i and \mathbf{H}_j for “seen” bindings from sorted access sa_i and sorted access sa_j , respectively.

Input : Query Q , result size k , and top- k test threshold τ .

Output : Approximated top- k result.

```

1 begin
2    $\beta \leftarrow \infty$ ,  $\omega \leftarrow -\infty$ 
3    $\mathcal{PC}.\text{initialize}()$  // initialize prior distributions
4   while  $|\mathbf{O}| < k$  or  $\min_{\bar{b} \in \mathbf{O}} \text{score}_Q(\bar{b}) < \beta$  do
5      $i \leftarrow \mathcal{PS}.\text{input}()$ 
6      $b \leftarrow$  next partial binding from sorted access  $sa_i$ 
7      $\beta \leftarrow \mathcal{BS}.\text{update}(b)$ 
8     // top- $k$  test, see Algorithm 10
9     if  $\mathcal{PC}.\text{probabilityTopK}(b, \omega) > \tau$  then
10       $\mathbf{O} \leftarrow \mathbf{H}_j \bowtie \{b\}$ 
11       $b \cup \mathbf{H}_i$  // add  $b$  to buffer  $\mathbf{H}_i$ 
12      if #new bindings  $\mathbf{B}$  in  $\mathbf{O} \geq$  training threshold then
13        // score distribution learning, see Algorithm 9
14         $\mathcal{PC}.\text{train}(\mathbf{B})$ 
15        Retain only  $k$  top-ranked bindings in  $\mathbf{O}$ 
16      if  $|\mathbf{O}| \geq k$  then // update smallest top- $k$  score  $\omega$ 
17         $\omega \leftarrow \min_{\bar{b} \in \mathbf{O}} \text{score}_Q(\bar{b})$ 
18      // return approximated top- $k$  results
19  return  $\mathbf{O}$ 

```

Pulling Strategy \mathcal{PS} and Bounding Strategy \mathcal{BS} . The A-PBRJ may exploit any bounding [55, 93, 116, 143] or pulling strategy [93, 117]. However, as the most common bounding strategy, we employ the *corner bound* [93] for our experiments (see Definition 15, p. 36).


Example 58

In example Figure 40-a, join j_1 currently has

$$\beta = \max\{8.5 + 0.9, 0.9 + 8.5\}$$

with $u_1 = l_1 = 8.5$, and $u_2 = l_2 = 0.9$.

Furthermore, we use the *corner-bound-adaptive pulling strategy* [93], which is defined in Definition 16, in our experiments.

 **Example 59**

For join j_1 in Figure 40-a, the corner-bound-adaptive pulling strategy may select either input, because

$$u_1 + l_2 = u_2 + l_1 \Leftrightarrow 8.5 + 0.9 = 0.9 + 8.5$$

and both inputs having two unseen partial bindings.

5.3.3.3 Probabilistic Component \mathcal{PC}

Given a partial binding b , we wish to know how likely b will contribute to the final top- k results. For this, we propose our new top- k test, which relies on two probabilities: (1) The probability that b contributes to a complete binding (*binding probability*). (2) The probability that complete bindings comprising b have higher scores than the current top- k bindings (*score probability*).

Binding Probability. To address the former probability, we use a selectivity estimation function. Intuitively, given a query Q , $\text{sel}(Q)$ estimates Q 's cardinality. That is, the expected number of Q 's results. Note, we introduced the selectivity estimation function in Definition 12.

 **Example 60**

Consider the query and data graph in Figure 39-a. Here, the selectivity of pattern $\text{tp}_3 = \langle m, \text{loc}, l \rangle$ is $\text{sel}(\text{tp}_3) = 2$, because two triples match this pattern.

Based on a selectivity estimation function, we define a *complete binding indicator* for a partial binding b :

$$\mathbb{1}(Q^u(b) \mid b) := \begin{cases} 1 & \text{if } \text{sel}(Q^u(b) \mid b) > 0 \\ 0 & \text{otherwise} \end{cases} \quad (21)$$


Intuitively, for a partial binding b , $\mathbb{1}(Q^u(b) \mid b)$ models *whether matching triples for b 's unevaluated patterns can exist, given variable assignments dictated by b .*

» **Definition 27**

Given a partial binding b and its unevaluated query patterns $Q^u(b)$ (see Definition 7, p. 22), we define $Q^u(b) \mid b$ as:

$$Q^u(b) \mid b := \{\overline{\text{tp}}_i\} \quad (22)$$

where $\text{tp}_i \in Q^u(b)$ and each variable v in tp_i that is bound by b is replaced with its assignment in b , $\mu_b(v)$, which results in a new pattern $\overline{\text{tp}}_i$.

 **Example 61**

Consider partial binding $b_{11} = (t_{11} = \langle m_3, \text{rating}, 8.5 \rangle, *, *)$ in Figure 40. Then, $\mathcal{Q}^u(b_{11}) \mid b_{11}$ is given by:

$$\mathcal{Q}^u(b_{11}) \mid b_{11} = \left\{ \underbrace{\langle m_3, \text{starring}, \text{"Audrey Hepburn"} \rangle}_{\overline{tp}_2}, \underbrace{\langle m_3, \text{loc}, l \rangle}_{\overline{tp}_3} \right\}$$

because variable m in pattern tp_2 and tp_3 is replaced with its assignment in b_{11} , $\mu_{b_{11}}(m) = m_3$. Further, it holds that

$$\mathbb{1}(\mathcal{Q}^u(b_{11}) \mid b_{11}) = 0$$

because selectivity for both patterns is 0.

It is important to notice that the complete binding indicator is independent of a specific selectivity function – *any selectivity estimation for BGP queries may be used*. With regard to text-rich Web data, our selectivity estimation approach in Chapter 4 may be employed.

We aimed at a simplistic baseline implementation of the complete binding indicator for our experiments. Therefore, we reused work from [127, 130] for a selectivity estimation function. The authors employed indexes for triple patterns with two constants: SP, PS, SO, OS, PO, and OP. Each index maps a $\langle \text{val}_1, \text{val}_2 \rangle$ pair to its cardinality, i.e., the number of its matching triples in the data. For instance, $\langle m_1, \text{starring} \rangle$ would map to 1 in Figure 39-a, since there is one triple matching the pattern $\langle m_1, \text{starring}, p \rangle$. However, the binding indicator only requires a selectivity estimation function to make a boolean decision: either $\text{sel}(\mathcal{Q}^u(b) \mid b) > 0$ or not. Thus, not all six indexes are necessary. In fact, indexes SP, PO, and SO are sufficient.

A simple implementation of $\text{sel}(\mathcal{Q}^u(b) \mid b)$ based on [127, 130] returns 1 iff

$$\forall \langle s, p, o \rangle \in \mathcal{Q}^u(b) \mid b, s \in \mathcal{V}_E : \text{SP.card}(\langle s, p \rangle) > 0 \quad \wedge \quad (23a)$$

$$\forall \langle s, p, o \rangle \in \mathcal{Q}^u(b) \mid b, o \in \mathcal{V}_E \uplus \mathcal{V}_A : \text{PO.card}(\langle p, o \rangle) > 0 \quad \wedge \quad (23b)$$


$$\forall \langle s, p, o \rangle \in \mathcal{Q}^u(b) \mid b, s \in \mathcal{V}_E \wedge o \in \mathcal{V}_E \uplus \mathcal{V}_A : \text{SO.card}(\langle s, o \rangle) > 0 \quad (23c)$$

and 0 otherwise, with $\text{card}()$ as cardinality function.

 **Example 62**

For $\mathcal{Q}^u(b_{11}) \mid b_{11}$ in Example 61, the binding indicator $\mathbb{1}(\mathcal{Q}^u(b_{11}) \mid b_{11})$ is 0, because the selectivity for both patterns is 0. Using above selectivity estimation implementation, probe in SP for $\langle m_3, \text{loc} \rangle$ returns “pair does not exist”. So, the function $\text{sel}(\mathcal{Q}^u(b_{11}) \mid b_{11})$ would correctly return 0.

Score Probability. For a partial binding b , let scores for bindings of b 's unevaluated patterns, $Q^u(b)$, be captured via a random variable $X_{Q^u(b)}^s$.

 **Example 63**

In Figure 40-a, partial binding b_{31} currently has a score of 1. However, scores for bindings to tp_1 and tp_2 are unknown and modeled via $X_{Q^u(b_{31})}^s$.

Then, we can obtain the probability for b contributing to a complete binding that has a score $\geq x$ as:

$$P\left(X_{Q^u(b)}^s \geq \delta(x, b)\right) \quad (24)$$

where $\delta(x, b) := x - \text{score}_Q(b)$. More precisely, partial binding b has a ‘‘certain’’ score, $\text{score}_Q(b)$, and only the score for its unevaluated patterns is unknown. So, $\delta(x, b)$ is the ‘‘delta’’ between b 's current score and a desired score x .

Top-k Test. Finally, we define the top-k test (see Algorithm 8, Line 8) via above probabilities. More precisely, we use the complete binding indicator to determine whether a partial binding b might contribute to a complete binding. Further, we employ the score probability to estimate how likely a complete binding, which comprises partial binding b , has a score that is larger than the smallest known top-k score, ω , see Algorithm 8 on Line 15:

$$\underbrace{\mathbb{1}(Q^u(b) \mid b)}_{\text{binding probability}} \cdot \underbrace{P(X_{Q^u(b)}^s \geq \delta(\omega, b))}_{\text{score probability}} > \tau \quad (25)$$

with $\tau \in [0, 1]$ as *top-k test threshold*.

Discussion.

- Threshold τ provides a key instrument for semantic search systems, since it allows to adjust the result accuracy. For instance, a system may decide to compute top-50 results in total and increase τ every time a new top-ranked binding can be reported. Generally speaking, τ should not be conceived as a constant, but rather as a function in reported top-ranked results. This way, systems can target typical end-user information needs, which do not require accurate low-ranked query results.
- The parameter ω refers to the smallest currently known top-k binding score. In particular, as long as no k complete bindings have been found, ω is set to $-\infty$ (see Algorithm 8, Line 2), and the score probability is always 1. So, a partial binding b is only pruned if it fails the complete binding indicator. That is, if b is not expected to contribute to any complete binding.
- Assume we have a tree of A-PBRJ operators and a partial binding b fails the complete binding indicator test at join j_i . Then, it is crucial to know, because of *which* pattern in $Q^u(b) \mid b$ the partial binding b fails the complete binding indicator test. In other words, we need to know which pattern $\bar{tp} \in Q^u(b) \mid b$ leads to $\mathbb{1}(Q^u(b) \mid b) = 0$.

	(a) Predictive Distributions	(b) Priors
Input i_1	$P(X_{i_1}^s)$ $Q^u = \{tp_2, tp_3\}$	$stat_2 \oplus stat_3:$ $(0.7 + 0.5, 0.12 + 0.08) = (1.2, 0.2)$
Input i_2	$P(X_{i_2}^s)$ $Q^u = \{tp_1, tp_3\}$	$stat_1 \oplus stat_3:$ $(8.1 + 0.5, 0.16 + 0.08) = (8.6, 0.24)$
Input i_3	$P(X_{i_3}^s)$ $Q^u = \{tp_1, tp_2\}$	$stat_1 \oplus stat_2:$ $(8.1 + 0.7, 0.16 + 0.12) = (8.8, 0.28)$
Input i_4	$P(X_{i_4}^s)$ $Q^u = \{tp_3\}$	$stat_3:$ $(0.5, 0.08)$

Table 5: (a) Given joins in Figure 40-a, we train four predictive score distributions (one for each input). For instance, $X_{i_1}^s$ models scores for bindings of $tp_2 \bowtie tp_3$. (b) Priors are based on sufficient statistics in Figure 40-b. We employ a summation as aggregation function \oplus in Figure 39-c. So, $stat_1 \oplus stat_3 = (8.1 + 0.5, 0.16 + 0.08) = (8.6, 0.24)$ etc.

Let \overline{tp}_k be that pattern and let join j_k be the join, which joins pattern tp_k with the remainder of the query. Then, we update all β thresholds associated with joins, which are “above” join j_i and “below” j_k in the tree. More precisely, we update β with the expected score of the partial binding b in that particular join.

This updating is necessary, because the partial binding b is assumed to successfully join with other bindings in joins “above” join j_i and “below” j_k . Only at join j_k , due to pattern \overline{tp}_k , binding b is assumed to fail.

- Last, each top-k test causes costs in the form of probability computations. We will provide an empirical estimation for those costs in our evaluation, see Section 5.3.4.3. However, recent work introduced a cost-aware rank join, which schedules sorted and random accesses based on their associated costs [117]. This work can be directly applied here. In fact, the top-k test may be treated as “one more” access in their optimization problem [117].

5.3.3.4 Score Distribution Learning

Distributions for random variables $X_{Q^u(b)}^s$ may be obtained by learning a score distribution $P(X_i^s)$ for each join input i . Note, partial bindings, which come from the same input, have the same set of unevaluated triple patterns. Thus, X_i^s captures scores of the unevaluated patterns from its partial bindings.

Example 64

In Figure 40-a, all partial bindings from input i_1 have $Q^u = \{tp_2, tp_3\}$ as unevaluated patterns. Thus, it holds that

$$P\left(X_{Q^u(b_{11})}^s\right) = P\left(X_{i_1}^s\right)$$

because binding b_{11} is produced by input i_1 . In fact, all bindings from input i_1 follow the same distribution, $P(X_{i_1}^s)$, which captures scores of bindings to $tp_2 \bowtie tp_3$.

Overall, for the running example, we need to learn the four distributions depicted in Table 5: $P(X_{i_1}^s)$, $P(X_{i_2}^s)$, $P(X_{i_3}^s)$, and $P(X_{i_4}^s)$.

Since we assume user/query-dependent ranking functions, we do not know the true distribution for X_i^s . A reasonable assumption is to use a *Gaussian distribution* for X_i^s and to employ a conjugate prior to train its unknown mean and variance, respectively.

As shown in [83], the mean of the Gaussian distribution for X_i^s also follows a Gaussian distribution, see Equation 26b. The variance of the Gaussian distribution for X_i^s follows an inverse-Gamma distribution, see Equation 26c. Hyperparameters $\alpha_0 = (\mu_0, \eta_0, \sigma_0^2, \nu_0)$ parameterize both distributions, where μ_0 is the prior mean with quality η_0 and σ_0^2 is the prior variance with quality ν_0 [83]:

$$X_i^s \sim \text{normal}(\mu, \sigma^2) \quad (26a)$$

$$\mu \mid \sigma^2 \sim \text{normal}\left(\mu_0, \frac{\sigma^2}{\eta_0}\right) \quad (26b)$$

$$\sigma^2 \sim \text{inverse-gamma}\left(\frac{\nu_0}{2}, \frac{\nu_0 \sigma_0^2}{2}\right) \quad (26c)$$

Prior Distribution. We initialize the prior distribution in Line 3 in Algorithm 8. More specifically, we specify a prior distribution for X_i^s via prior hyperparameters α_0 for each input i . For α_0 , we require sufficient score statistics in the form of a sample mean

$$\bar{x} = \frac{1}{n} \sum_{x_i \in \mathbf{x}} x_i$$

and a sample variance

$$s^2 = \frac{1}{(n-1)} \sum_{x_i \in \mathbf{x}} (x_i - \bar{x})^2$$

with \mathbf{x} as sample. There are multiple ways to obtain the necessary score samples:

Example 65

Figure 40-b depicts three sufficient statistics based on information from the sorted accesses:

- ① With regard to sorted access sa_1 , we rely on offline information to obtain the sufficient statistics. Recall, rating scores are known before runtime. Therefore, $\bar{x}_1 = 8.1$ and $s_1^2 = 0.16$ can be computed offline.
- ② Online information is exploited for sorted access sa_2 . Here, the list of matching triples for keywords “Audrey” and “Hepburn” must be fully materialized. So, $\bar{x}_2 = 0.7$ and $s_2^2 = 0.12$ can be computed from scores of triples in this list.

- ③ Last, given access sa_3 , we neither have offline scores, nor a fully materialized list of triples (sa_3 loads a triple solely upon a pull request). In lack of more information, we assume each score to be equal likely, i.e., a uniform distribution. Given a min. score of 0 and a max. score of 1, we get: $\bar{x}_3 = 0.5$ and $s_3^2 = 0.08$.

Algorithmus 9 : $\mathcal{PC}.train()$

Param. : Weight $w \geq 1$ for score sample \mathbf{x} .
Buffer : Buffer \mathbf{A} storing hyperparameters α .
Input : Complete bindings $\mathbf{B} \subseteq \mathbf{O}$ and join j .

```

1 begin
  // train hyperparameters for each input
2  foreach input  $i$  in join  $j$  do
    // load prior hyperparameters for input  $i$ 
3     $\alpha_n = (\mu_n, \eta_n, \sigma_n^2, \nu_n) \leftarrow \mathbf{A}_i$ 
    // get scores of bindings for input  $i$ 's unevaluated patterns
4    foreach complete binding  $b \in \mathbf{B}$  do
5      get binding  $b'$  comprised in  $b$ , which matches unevaluated
      patterns
6      add  $score_Q(b')$  to score sample  $\mathbf{x}$ 

    // compute sample mean and variance
7     $\bar{x} \leftarrow \text{mean}(\mathbf{x}) = n^{-1} \sum x_i$ 
8     $s^2 \leftarrow \text{var}(\mathbf{x}) = \frac{1}{(n-1)} \sum (x_i - \bar{x})^2$ 

    // compute posterior hyperparameters
9     $\nu_{n+1} \leftarrow \nu_n + w, \quad \eta_{n+1} \leftarrow \eta_n + w$ 
10    $\mu_{n+1} \leftarrow \frac{1}{\eta_{n+1}} \cdot (\eta_n \mu_n + w \bar{x})$ 
11    $\sigma_{n+1}^2 \leftarrow \frac{1}{\nu_{n+1}} \cdot \left( \nu_n \sigma_n^2 + (w-1)s^2 + \frac{\eta_n w}{\eta_{n+1}} \cdot (\bar{x} - \mu_n)^2 \right)$ 

    // store new (posterior) hyperparameters for input  $i$ 
12    $\mathbf{A}_i \leftarrow \alpha_{n+1} = (\mu_{n+1}, \eta_{n+1}, \sigma_{n+1}^2, \nu_{n+1})$ 

```

We initialize hyperparameters α_0 with μ_0 as sample mean, σ_0^2 as sample variance, and $\eta_0 = \nu_0$ as sample quality. For every input, we aggregate necessary sample means/variances for μ_0/σ_0^2 . For example, given input i_1 with unevaluated patterns $Q^u = \{tp_2, tp_3\}$, we sum up (aggregate) statistics $stat_2$ and $stat_3$: $\bar{x}_2 + \bar{x}_3$ for μ_0 and $s_2^2 + s_3^2$ for σ_0^2 , see Table 5.

Hyperparameters η_0 and ν_0 are used to quantify the prior quality. For instance, $stat_1$ and $stat_2$ are exact statistics, while $stat_3$ relies on a uniform distribution. So, weighting reflects the prior's accuracy.

Posterior Distribution. Having estimated a prior distribution, *we continuously update this distribution with scores seen during query processing.*

Intuitively, each time new complete bindings are produced, all prior distributions could be trained – as illustrated in Algorithm 9. That is, complete binding

scores are used to update hyperparameters from the previous n -th training iteration, α_n , resulting in new posterior hyperparameters, α_{n+1} . For this, we use standard training on Lines 10-11 [83]:

$$\mu_{n+1} = \frac{\eta_n \mu_n + w \bar{x}}{\eta_{n+1}} \quad (27a)$$

$$\sigma_{n+1}^2 = \frac{1}{\nu_{n+1}} \cdot \left(\nu_n \sigma_n^2 + (w-1)s^2 + \frac{\eta_n w}{\eta_{n+1}} \cdot (\bar{x} - \mu_n)^2 \right) \quad (27b)$$

In simple terms, the prior mean μ_n is updated with the new sample mean \bar{x} (see Equation 27a and Line 10). Furthermore, the prior variance σ_n^2 is updated with the sample variance s^2 (see Equation 27b and Line 11). To obtain the sample mean and sample variance, each input computes its own score sample \mathbf{x} (see Lines 5-6). This is necessary since every X_i^s models scores for different unevaluated patterns.

Prior hyperparameters are weighted via η_n and ν_n . Further, for each hyperparameter update, a parameter w is used as a weight indicating the quality of sample \mathbf{x} . Finally, new hyperparameters α_{n+1} are stored on Line 12.

Example 66

Consider input i_1 in Table 5 and assume $\eta_0 = \nu_0 = 1$. Then, the prior is: $\alpha_0 = (1.2, 1, 0.20, 1)$. We observe scores $\mathbf{x} = \{x_1, x_2\}$ from

$$\mathbf{B} = \{\langle t_{12}, t_{21}, t_{31} \rangle, \langle t_{13}, t_{22}, t_{32} \rangle\}$$

with $w = |\mathbf{x}| = 2$ and

$$x_1 = 1.9 = \text{score}_Q(t_{21}) + \text{score}_Q(t_{31})$$

$$x_2 = 0.9 = \text{score}_Q(t_{22}) + \text{score}_Q(t_{32})$$

So, we have $s^2 = 0.5$ and $\bar{x} = 1.4$, which leads to the posterior hyperparameters:

$$\eta_1 = \nu_1 = 1 + 2 = 3$$

$$\sigma_1^2 = \frac{1}{3} \cdot \left(0.2 + (2-1) \cdot 0.5 + \frac{(1.4-1.2)^2}{3} \right) = 0.71$$

$$\mu_1 = \frac{(1.2 + 2 \cdot 1.4)}{3} = 1.33$$

After each such update only posterior hyperparameters are stored, thereby making the learning highly space and time efficient:

◆ Theorem 5: Distribution Learning Space Complexity

Given an A-PRBJ operator, at any time during query processing, we require a space complexity of $O(1)$ for score distribution learning.

Sketch of Proof

Given an A-PRBJ operator, every of its inputs i stores only a parameter vector, hyperparameters α , during each training iteration (Algorithm 9, Line 12). Since each vector α has a fixed size, the space consumption remains constant. In particular, vector α is independent of the number of training iterations ■

For the learning time complexity we can show:

•• Theorem 6: Distribution Learning Time Complexity

Given an A-PRBJ operator, a query Q , and \mathbf{B} complete bindings for Q , score learning time complexity is bounded by $O(|\mathbf{B}|)$.

Sketch of Proof

Given an A-PRBJ operator and a set of complete bindings \mathbf{B} : A score sample, \mathbf{x} , is constructed (Algorithm 9, Lines 5-6) with $O(|\mathbf{B}|)$ complexity. Mean and variance is computed from \mathbf{x} in $O(|\mathbf{x}|)$ time. However, since $|\mathbf{x}| \leq |\mathbf{B}|$, it holds that $O(|\mathbf{x}|) \in O(|\mathbf{B}|)$. In fact, computation of mean and variance could also be done while collecting the sample (Algorithm 9, Lines 5-6). Further, hyperparameters are updated via \mathbf{x} in constant time (Algorithm 9, Lines 9-11). Overall, the training has a complexity of: $O(|\mathbf{B}|)$ ■

Algorithmus 10 : \mathcal{PC} .probabilityTopK()

Buffer : Buffer \mathbf{A} storing hyperparameters.
Input : Partial bindings b , input i , and join j .
Output : Probability that b will result in one (or more) final top-k bindings.

```

1 begin
   // load hyperparameters  $\alpha_n$  for input  $i$ 
2    $\alpha_n = (\mu_n, \eta_n, \sigma_n^2, \nu_n) \leftarrow \mathbf{A}_i$ 
   // posterior predictive distribution in closed-form as
   // Student's t-distribution based on hyperparameters  $\alpha_n$ 
3    $X_i^s \sim t_{(\nu_n)} \left( x \mid \mu_n, \frac{\sigma_n^2(\eta_n+1)}{\eta_n} \right)$ 
   // compute score probability
4    $p_S \leftarrow P \left( X_{Q^u(b)}^s \geq \delta(\omega, b) \right) = P \left( X_i^s \geq \delta(\omega, b) \right)$ 
   // compute binding probability
5    $p_B \leftarrow \mathbb{1}(Q^u(b) \mid b)$ 
   // probability that  $b$  contributes to top-k results
6   return  $p_S \cdot p_B$ 

```

Predictive Distribution. We provide an implementation of the top-k test in Algorithm 10. At any point during query processing, one may need to perform this test. For this, our approach allows to always give a distribution for X_i^s , based on the currently known hyperparameters α_n , see Line 2. Since hyperparameters are continuously trained, the distribution quality improves over time.

More specifically, we use the *posterior predictive distribution*. This distribution estimates probabilities for *new* scores, based on observed scores and the prior distribution. For a Gaussian conjugate prior, this distribution can be easily obtained in a closed form as non-standardized Student's t-distribution with ν_n degrees of freedom [83], see Line 3.

We compute $P(X_{Q^u(b)}^s) = P(X_i^s)$ by means of the posterior predictive distribution on Line 4. Last, we compute the binding probability via a selectivity estimation function on Line 5 and return b 's top-k test probability in Line 6.

5.3.3.5 Discussion

Refined Conjugate Priors. Whenever we have offline information about the true distribution of $X_{Q^u(i)}^s$ (or good approximation for it), we can replace the Gaussian conjugate prior in Equation 26 with a more suitable one. For this, only minor changes in the score distribution training and the predictive distribution estimation are required. No further modifications are needed – the top-k test works with any valid score distribution for $X_{Q^u(i)}^s$.

A wide variety of discrete/continuous conjugate priors are known. Thus, in the best case, there is a conjugate prior for the true distribution of $X_{Q^u(i)}^s$. If no suitable conjugate prior exists, we can exploit a mixture of multiple conjugate priors:

$$\sum_i w_i P_i(\Theta)$$

with each $P_i(\Theta)$ being a conjugate prior and w_i as weights such that $\sum_i w_i = 1$ and $0 < w_i < 1$. In particular, it has been shown that any distribution from the exponential family can be approximated (arbitrarily close) by means of a mixed conjugate prior [22].

Maintenance. We require maintenance of binding/score probabilities. Binding probabilities are estimated via a selectivity estimation function. Maintenance of these statistics varies with the specific selectivity estimation implementation. For instance, we outlined maintenance of our selectivity estimation approach for hybrid queries over Web data in Section 4.3.2.3.

We train score distributions throughout query processing. So, only for prior distributions sufficient statistics are necessary and must be maintained. This maintenance differs, depending on the ranking functions employed:

- For user-/query-dependent ranking functions (e.g., the distance ranking constraint in Figure 39-b) scores are unknown before runtime. Thus, no sufficient statistic can be stored/maintained. However, in such a case, a minimal and maximal score must be kept. For instance, for the distance ranking constraint, we would store a minimal and maximal score as 0 and

1, respectively. This way, we can assume a uniform score distribution as naïve prior, and compute mean and variance as:

$$\text{stat}_3 = (\bar{x}_3 = 0.5, s_3^2 = 0.08)$$

see Figure 40-b and Example 65.

- With regard to user-/query-independent ranking functions (e.g., the rating ranking constraint in Figure 39-b) sufficient statistics can be computed at indexing time. That is, one should maintain a sample mean and a sample variance (see Example 65). In fact, one may even store further distribution characteristics, e.g., distribution skewness or symmetry. This way, more refined conjugate priors could be estimated (see paragraph above).

5.3.3.6 Theoretical Analysis

In this section, we present a theoretical analysis with regard to the *effectiveness* of the A-PBRJ operator. More precisely, we discuss the quality of the learned score distributions in Theorem 7 and Theorem 8, and provide bounds for the approximation error in Theorem 9.

Distribution Quality. The aggregation function \oplus , which is used by the ranking function score_Q , could be any monotonic function.

However, when we restrict the aggregation to a summation, we can formally show that a Gaussian distribution/conjugate prior is a *good approximation for the true distribution* of $X_{Q^u(i)}^s$. Notice, many common aggregations employ summations, e.g., TF-IDF inspired functions can be represented by summations [157]. For such a summation-based aggregation function it holds:

• Theorem 7

Given a query $Q = \{tp_k\}_k$ and $X_{Q^u(i)}^s = \sum_{tp_k \in Q^u(i)} X_{tp_k}^s$, the Central Limit Theorem (CLT) holds:

$$\frac{\sum_k (X_{tp_k}^s - \mu_k)}{\sqrt{\sum_k \sigma_k^2}} \underset{n \rightarrow \infty}{\sim} \text{normal}(0, 1)$$

with n as the number of patterns in $Q^u(i)$, and $X_{tp_k}^s$ as random variable modeling scores of bindings for pattern tp_k . Further, μ_k and σ_k^2 stand for the finite mean and variance of $X_{tp_k}^s$, respectively.

Sketch of Proof (Informal)

Since we do not have knowledge about the ranking function, score_Q , or the distribution for $X_{tp_k}^s$, we can only outline a very informal sketch of proof in the following. Our argumentation is based on two aspects:

- ① Recall, we define a *separate* ranking function score_Q for every triple pattern tp_k in query Q , see Definition 13, p. 32. In particular, each function computes its score solely by considering the partial binding of “its own” pattern, tp_k . So, it is reasonable to assume that scores for bindings of two patterns, tp_i and tp_j , in query Q are independent. This leads to random variables $X_{\text{tp}_i}^s$ and $X_{\text{tp}_j}^s$ being independent:

$$X_{\text{tp}_i}^s \perp X_{\text{tp}_j}^s$$

- ② Without loss of generality, we assume each random variable $X_{\text{tp}_k}^s$ to have a finite mean μ_k and variance σ_k^2 . Notice, most common distributions feature a finite mean and variance.

In its simplest form, the Central Limit Theorem is only applicable to i.i.d. random variables [68]. However, in the Lindeberg Theorem this restriction is lifted. That is, every variable $X_{\text{tp}_k}^s$ may adhere to a different distribution [68]:

→ **Lemma 2: Lindeberg Condition [68]**

If

$$\lim_{n \rightarrow \infty} \frac{1}{s_n^2} \sum_k \mathbb{E} \left((X_{\text{tp}_k}^s - \mu_k)^2 \cdot \mathbb{1}(|X_{\text{tp}_k}^s - \mu_k| \geq \varepsilon s_n) \right) = 0$$

holds, where $s_n^2 = \sum_k \sigma_k^2$, then the *Central Limit Theorem* in Theorem 7 holds.

Further, it is known that [23]:

→ **Lemma 3**

If each random variable $X_{\text{tp}_k}^s$ is uniformly bounded, and $\lim_{n \rightarrow \infty} s_n = \infty$, then the Lindeberg Condition in Lemma 2 holds.

Sketch of Proof

Since our score_Q function is bounded in $[0, 1]$ (defined in Definition 13, p. 32), every variable $X_{\text{tp}_k}^s$ is also bounded: $P(0 \leq X_{\text{tp}_k}^s \leq 1) = 1$.

Further, the variance σ_k^2 can be expected to be > 0 for each $X_{\text{tp}_k}^s$, because $X_{\text{tp}_k}^s$ models ranking scores. That is, ranking scores are supposed to vary between results, in order to assist users in differentiating between results, and quickly discover results of interest. Therefore, it can be expected that $s_n = \sum_k \sigma_k^2 \rightarrow \infty$ with $n \rightarrow \infty$ ■

In simple terms, Theorem 7 states that the true distribution of $X_{Q^u(i)}^s$ converges (in the number of patterns in $Q^u(i)$) to a Gaussian distribution.

However, the next question is: “how fast” does $X_{Q^u(i)}^s$ converge to a Gaussian distribution? In other words, “how large” must $Q^u(i)$ be, in order for $X_{Q^u(i)}^s$ to follow a Gaussian distribution? For this convergence it holds:

•• **Theorem 8: Berry-Esseen Theorem [68]**

Let $\rho_k = E(|X_{tp_k}^s|^3) < \infty$ be the third absolute normalized moment of $X_{tp_k}^s$. Then, it holds [68]:

$$\sup_x |F(x) - \phi(x)| \leq C \cdot \frac{\sum_k \rho_k}{(\sum_k \sigma_k^2)^{\frac{3}{2}}}$$

with $\phi(x)$ as standard Gaussian CDF, and $F(x)$ as exact CDF of $X_{Qu(i)}^s$. Further, tp_k , μ_k , and σ_k^2 are defined as in Theorem 7.

C is a constant in Theorem 8, which is currently estimated as $0.4097 \leq C \leq 0.56$ [68]. Intuitively, Theorem 8 gives an *absolute bound* for the difference between the true distribution of $X_{Qu(i)}^s$ and a Gaussian distribution.

Approximation Error. Let X_i^e denote a random variable for the error introduced by pruning from input i . This error may be measured as the number of pruned partial bindings from i , which would have contributed to the final top- k result. Then, it holds that:

→ **Lemma 4**

Random variable X_i^e follows a binomial distribution such that:

$$X_i^e \sim \text{bin}(c_i, \varepsilon + \tau) \quad (29)$$

c_i stands for the the number of bindings pulled from input i , in order to produce the top- k results. Further, τ is the error threshold from Equation 25, and ε is a small constant ≥ 0 .

Sketch of Proof

From a given input i , let's assume that we materialize c_i partial bindings. Every such binding could be pruned "wrongfully" by the top- k test in Equation 25, either because the binding probability was falsely estimated as 0 or because the score probability was smaller than the threshold τ .

Let the probability for the former be bounded by a constant $\varepsilon \geq 0$, while the probability for the latter is known to be $\leq \tau$. Thus, the overall probability for a partial bindings to be wrongfully pruned in Equation 25 is $\leq \varepsilon + \tau$.

Further, pulling c_i partial bindings from input i may be conceived as c_i trails, where a wrongly pruned binding is a "hit". Therefore, we can model X_i^e by means of binomial distribution, with c_i as number of trails and $\varepsilon + \tau$ as "success" probability. Overall, it holds that: $X_i^e \sim \text{bin}(c_i, \varepsilon + \tau)$ ■

Note, ε is a small error introduced by the binding indicator function. This error depends on the accuracy of the selectivity estimation. However, as the binding indicator only requires a binary decision, its induced error is frequently very small. In fact, our simplistic implementation in Equation 23 is exact: $\varepsilon = 0$.

Given a tree of A-PBRJ operators, which have a total of n inputs, let X^e capture the overall error. That is, random variable X^e models the total number of wrongly pruned partial bindings. We can show that X^e also adheres to a binomial distribution:

• Theorem 9

Based on Lemma 4, it holds that:

$$X^e \sim \text{bin} \left(\sum_{i=1}^n c_i, \varepsilon + \tau \right) \quad (30)$$

with c_i , τ , and ε , as defined in Lemma 4.

Sketch of Proof

Given a tree of joins having n inputs: $\{i_1, \dots, i_n\}$. Let every input i_j pull c_j partial bindings, in order for the join tree to produce the desired k top-ranked results. Further, the error (the number of wrongly pruned partial bindings) for each input i_j is modeled via variable

$$X_i^e \sim \text{bin}(c_j, \varepsilon + \tau)$$

see Lemma 4.

False positives/negatives results comprised in the final top- k bindings are caused by wrongly pruned partial bindings. More precisely, for a given input i_j , every wrongly pruned partial binding could lead to a false positive/negative top- k result. Thus, errors from the individual inputs “sum up” to a total error – captured by X^e .

In other words, random variable X^e is a summation over the random variables X_i^e . Further, errors made in the inputs are independent of each other. That is, every pair of variables, X_i^e and X_j^e , is independent: $X_i^e \perp X_j^e$.

Thus, X^e is again a binomial distribution with $\sum_{i=1}^n c_i$ trials and “success” probability $\varepsilon + \tau$: $X^e \sim \text{bin}(\sum_{i=1}^n c_i, \varepsilon + \tau)$ ■

Finally, we can give the expected error as a function in threshold τ :

$$E(X^e) = \sum_{i=1}^n c_i \cdot (\varepsilon + \tau) \quad (31)$$

	SP ² Queries	DBPSB Queries
# Queries	13	120
# Triple patterns	2-9	2-4
Mean(# Triple patterns)	5	2.8
Var(# Triple patterns)	6.4	0.6
# Results	1 - 5.4E ⁶	1 - 50
Mean(# Results)	545E ³	3.9
Var(# Results)	2.1E ¹²	51.6

Table 6: Query statistics for the SP² and DBPSB benchmark.

5.3.4 Evaluation

We conducted experiments for (1) *analyzing the efficiency and effectiveness of the A-PBRJ operator*, and (2) *inspecting the behavior of our probabilistic component PC*. By means of the former, we illustrate the overall performance of our approach, when compared with the exact PBRJ. The latter provides insights into overhead and accuracy of the probabilistic component.

5.3.4.1 Evaluation Setting

Benchmarks. We used two SPARQL benchmarks: (1) The SP² benchmark featuring synthetic DBLP data [141]. (2) The DBpedia SPARQL benchmark (DBPSB), which holds real-world DBpedia data and queries [123]. For both benchmarks, we generated datasets with 10M triples.

We translated the benchmark queries to our query model. Queries featuring no triple patterns could not be translated – we omitted 12 and 4 queries in DBPSB and SP², respectively. We generated DBPSB queries as proposed in [123]: Overall, we used 8 seed queries with 15 random bindings, which led to a total of 120 DBPSB queries. For SP² we employed 13 queries. In total, we had a comprehensive load of 133 queries. Query statistics are given in Table 6 and a complete query listing is shown in Section A.3.

Systems. We randomly generated bushy query plans. For a given query, all systems rely on the same plan. We implemented three systems that solely differ in their join operator:

- A system with *join-sort* operator, JS, which *does not employ top-k processing*, but instead produces all results and then sorts them.
- An *exact and complete* top-k join operator, PBRJ, featuring the corner-bound in Definition 15 and the corner-bound-adaptive pulling strategy in Definition 16. PBRJ is an implementation of Algorithm 1 and resembles previous approaches for top-k processing over RDF data [115, 2].

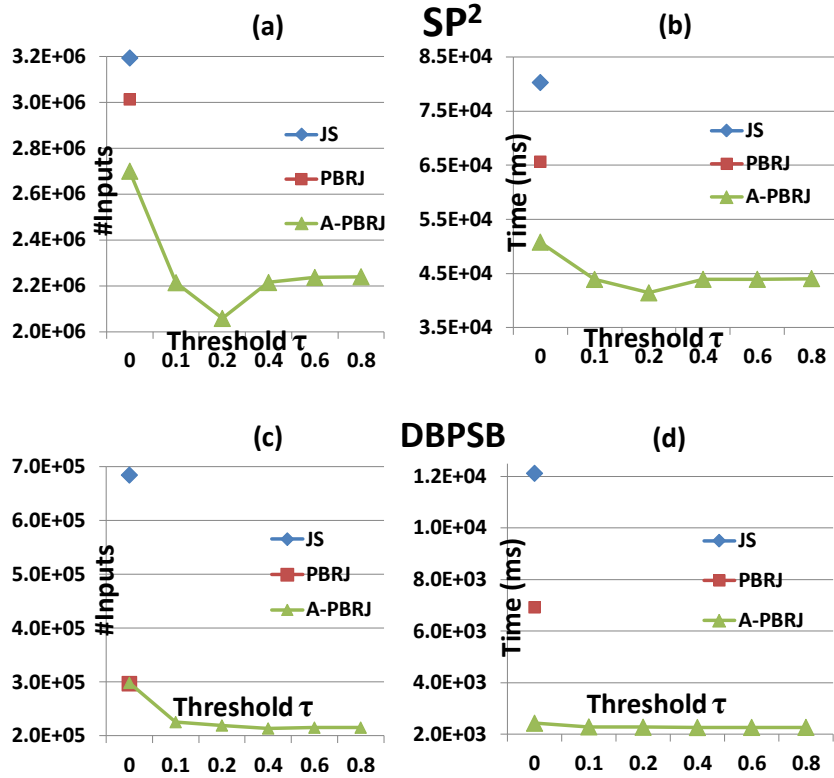


Figure 41: Efficiency evaluation results for SP²/DBPSB: (a)/(c) number of inputs versus threshold τ . (b)/(d) time versus threshold τ .

- Last, we implemented our *approximate* top-k join operator, A-PBRJ, illustrated in Algorithm 8.

Score learning and top-k test implementation for the A-PBRJ operator follows Algorithm 9 and Algorithm 10, respectively. We employed a training threshold of 10 bindings for score distribution learning, see Line 11 in Algorithm 8. Further, we used sufficient statistics based on a uniform distribution over $[0, 1]$, as discussed in Example 65 for sorted access sa_3 . Prior weights ν_0 and η_0 are both 1, see Algorithm 9. Weight w in Algorithm 9 is the sample size, $|x|$. Scores for single triple pattern bindings are random (see below) and complete binding scores are computed as summation. We reused the selectivity estimation implementation from [127, 130] for our binding probabilities. The complete binding indicator implementation adheres to Equation 23.

We implemented all systems in Java 6. Experiments were run on a Linux server with two Intel Xeon 5140 CPUs at 2.33GHz, 48GB memory (16GB assigned to the JVM), and a RAID₁₀ with IBM SAS 148GB 10K rpm disks. Before each query execution, all operating system caches were cleared. The presented values are averages collected over five runs.

Parameters. We employed multiple parameters, in order to examine the behavior of above systems with regard to different settings:

- We vary the number of results to be computed: $k \in \{1, 5, 10, 20\}$.

- We chose triple pattern binding scores, $\text{score}_Q(t)$, at random with distribution $d \in \{u, n, e\}$ (uniform, normal, and exponential distribution). By means of varying distributions, we aim at an abstraction from a particular ranking function and examine performance for different “classes” of functions. We employed standard parameters for all distributions and normalized scores to be in $[0, 1]$.
- Last, we used top-k test thresholds $\tau \in [0, 0.8]$ for inspecting the trade-off between computation efficiency and effectiveness.

Metrics. We rely on the following metrics to measure efficiency and effectiveness aspects of the systems. As efficiency metrics we use: (1) the number of inputs processed and (2) the time needed for result computation.

As effectiveness metrics we use: (1) Precision: fraction of approximated top-k results that are exact top-k results. (2) Recall: fraction of exact top-k results, which are reported as approximate top-k results. Notice, precision and recall have identical values, since both share the same denominator k . We therefore discuss only precision results in the following. Further, precision is given as average over our query load (so-called macro-precision). (3) Last, we employ the score error defined in [157], which compares the approximate versus exact top-k score as:

$$\frac{1}{k} \sum_{b=1, \dots, k} |\text{score}_Q^*(b) - \text{score}_Q(b)|$$

with $\text{score}_Q^*(b)$ and $\text{score}_Q(b)$ as approximated and exact score for binding b , respectively.

5.3.4.2 Evaluation Results: A-PBRJ

In the next paragraphs, we inspect the overall behavior of our A-PBRJ approach versus the baseline systems, which compute exact and complete results.

Efficiency Results. Efficiency results are depicted in Figure 41-a/b (c/d) for SP² (DBPSB). We observed A-PBRJ to save inputs and computation time. For SP² (DBPSB), A-PBRJ needed up to 25% (23%) less inputs versus baseline PBRJ and 30% (67%) versus JS. We explain these gains with pruning of partial bindings via our top-k test, thereby omitting “unnecessary” joins and join attempts. In fact, we were able to prune up to 40% (90%) of the inputs, given SP² (DBPSB). Fewer inputs translated to time savings of 35% (65%) versus PBRJ and 47% (80%) versus JS, given SP² (DBPSB).

Interestingly, we saw an increase in inputs for $\tau \in [0.2, 0.4]$ in SP² and $\tau \in [0.4, 0.8]$ in DBPSB, see Figure 41-a/c. For instance, comparing $\tau = 0.2$ and $\tau = 0.4$ in SP², A-PBRJ read 8% more inputs. DBPSB was less affected: we noticed a marginal increase of 2% for $\tau = 0.4$ versus $\tau = 0.6$. We explain the increase in both benchmarks with a too “aggressive” pruning – too many partial bindings were pruned wrongfully. That is, many pruned bindings would have led to a larger or even a complete binding. In turn, this led to more inputs being read, in order to produce the desired k results. In fact, $\tau \in [0.6, 0.8]$ was even more

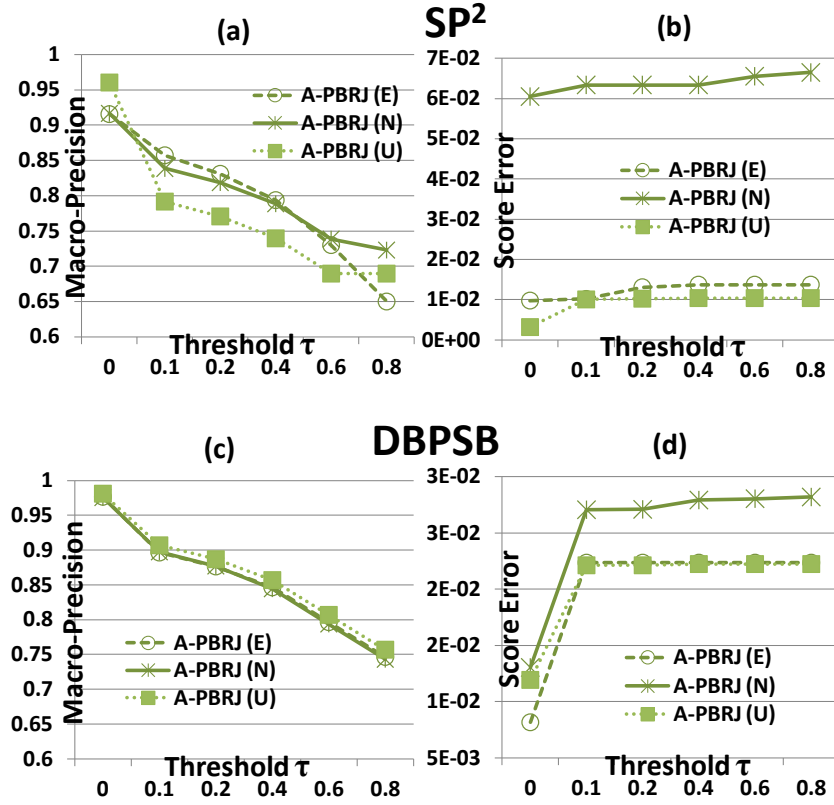


Figure 42: Effectiveness evaluation results for SP²/DBPSB: (a)/(c) macro-precision versus threshold τ . (b)/(d) score error versus threshold τ .

aggressive. However, the ratio between pruned bindings and read inputs was high enough to compensate for the extra inputs. Overall, we saw a “sweet spot” at $\tau \approx 0.2$ for SP² and DBPSB. Here, we noted pruning to be fairly accurate, i.e., only few partial bindings were wrongfully pruned. In fact, we observed high precision (recall) values for both benchmarks given $\tau \approx 0.2$: 80% (88%) in SP² (DBPSB) – as discussed below. With regard to computation time for SP² and DBPSB queries, we noticed similar effects as for the inputs, see Figure 41-b/d. In particular, the “sweet spot” at $\tau \approx 0.2$ is also reflected here.

As expected, we observed inputs and time to increase in k for A-PBRJ and PBRJ. For instance, comparing $k = 1$ and $k = 20$, A-PBRJ needed a factor of 1.2 (5.7) more time, given SP² (DBPSB). Similarly, 1.2 (6.8) times more inputs were consumed by A-PBRJ for SP² (DBPSB). We explain this behavior with more inputs/join attempts being required to produce a larger result. PBRJ leads to a similar performance decrease. For instance given $k = 1$ versus $k = 20$ in SP², PBRJ needed a factor of 1.3 (1.2) more inputs (time). Note, as baseline JS simply computed all results, this system was not affected by k .

Furthermore, we could not find a correlation between system performance and score distributions. In other words, score distributions (ranking functions) had no impact on A-PBRJ’s performance. For instance given DBPSB queries, A-PBRJ resulted in the following gains versus PBRJ with regard to inputs (time): 27% (65%) for e distribution, 23% (64%) given u distribution, and 21% (64%) for n distribution.

Last, with regard to parameter τ , we noted A-PBRJ’s efficiency to increase with $\tau \in [0, 0.2]$, given SP² and DBPSB. However, as outlined above, too aggressive pruning led to “inverse” effects. An important observation is, however, that our approach was already able to achieve performance gains with a very small $\tau < 0.1$. Here, partial bindings were pruned primarily due to their low binding probability. In fact, A-PBRJ could even save time for $\tau = 0$: 26% (60%) with SP² (DBPSB). We inspected queries leading to such saving and saw that many of their partial bindings had a binding probability ≈ 0 . We argue that this is a strong advantage of A-PBRJ: *even for low error thresholds (leading to a minor effectiveness decrease), we could achieve efficiency gains.*

Effectiveness Results. In the following, we analyze A-PBRJ in terms of its accuracy. Note, baselines PBRJ and JS *always compute exact and complete results*. So, we restrict our attention to the A-PBRJ system.

Figure 42-a/c (b/d) depicts the macro-precision (score error) for varying score distributions. We observed high precision values of up to 0.98 for both benchmarks, see Figure 42-a/c. Given $\tau < 0.1$, all distributions led to very similar precision results $\in [0.8, 0.95]$ and $[0.90, 0.98]$ for SP² and DBPSB, respectively. In general, A-PBRJ’s effectiveness is hardly affected by a particular score distribution, see Figure 42-a/c. We explain these good approximations with accurate score/binding probabilities – no matter the score distribution.

Moreover, even for large $\tau \in [0.6, 0.8]$ A-PBRJ achieved a high macro-precision in $[0.75, 0.8]$ on DBPSB queries. This is because, DBPSB queries feature selective patterns and have only a small result cardinality ≤ 10 . Thus, “chances” of pruning a final top-k binding were quite small – even for a large τ . Moreover, A-PBRJ led to a very effective pruning via binding probabilities, as many partial bindings had a binding probability ≈ 0 (due to the high query selectivity). This way, A-PBRJ pruned up to 97% of the total inputs for some DBPSB queries.

In order to quantify “how bad” false positive/negative results are, we employed the score error metric, see Figure 42-b/d. For both benchmarks, we observed that score error was $\in [0.07, 0.11]$ for a small $\tau < 0.1$. We explain this with our high precision (recall). That is, A-PBRJ led to only few false positive/negative top-k results given $\tau < 0.1$. As expected, score error increased in τ , due to more false positives/negatives top-k results. Overall, however, score error results were very promising: we saw an average score error of 0.03 (0.02), given SP² (DBPSB).

With regard to parameter k , we observed that k does not impact A-PBRJ’s effectiveness. Given SP², we saw A-PBRJ to be fairly stable in different values for parameter k . For instance, macro-precision was in $[0.8, 0.85]$ as average over all k and $\tau = 0.1$. Also for the DBPSB benchmark, we noted only minor effectiveness fluctuations: macro-precision varied around 7% with regard to different k .

We noticed A-PBRJ’s effectiveness to not be influenced by varying score distributions, see Figure 42-a/b/c/d. Given SP², we saw a macro-precision of 0.79 for u distribution, 0.79 for e distribution, and 0.80 for n distribution. Also for the DBPSB benchmark, we observed only minor changes in macro-precision: 0.87 for u distribution, 0.85 for e as well as n distribution.

With regard to the effectiveness of A-PBRJ versus parameter τ , we noticed that metrics over both benchmarks decreased with increasing τ . For instance, macro-

	SP ²	DBPSB	Dist.	SP ²	DBPSB
Time (ms)	1 - 300	1 - 24	e	0.34	0.04
# Samples	1 - 4E ⁶	1 - 50	n	0.34	0.01
Avg. #Sample	400E ³	4	u	0.31	0.02

(a) Efficiency: average learning time and the number of learning samples. (b) Effectiveness: average p-value from the Kolmogorov-Smirnov test.

Table 7: Efficiency and effectiveness of score distribution learning.

precision decreased for $\tau = 0$ versus $\tau = 0.8$ with 27% (23%), given SP² (DBPSB). Such a behavior can be expected, since chances of pruning “the wrong” bindings increase with higher τ values. Thus, while leading to efficiency gains (discussed above), a higher value for τ causes effectiveness losses.

5.3.4.3 Evaluation Results: Probabilistic Component

In this section, we analyze the performance of the probabilistic component in terms of its efficiency and effectiveness. Since binding probabilities are estimated via a given selectivity estimation framework based on previous works [127, 130], we focus on the learning and the computation of score probabilities.

Efficiency Results. First, we analyze the overhead introduced by score distribution learning. For this, we measured the time needed for hyperparameter training, see Algorithm 9. We set the training threshold, i.e., the number of new bindings after which a new training procedure is triggered, to 1 (Algorithm 8, Line 11). Table 7-a gives average training times and the number of samples for distribution training.

We observed average learning times $\in [1, 300]$ ms ($[1, 24]$ ms) over all score distributions, queries and thresholds τ , given the SP² (DBPSB) benchmark. We noted the driving factor to be the overall query selectivity. That is, SP² queries often had a large cardinality $\gg 100$, which led to a high number (up to 4,227,732) of training samples. In contrast, DBPSB featured highly selective queries, resulting in few training iterations – only up to 50 score samples were available on average. Overall, this explains the additional training time (factor 12.5) needed for SP² queries, when compared to DBPSB. Note, one can easily cope with high cardinality queries by: (1) setting a larger training threshold or (2) stop the distribution learning, if the distribution “quality” does not improve any more. However, such optimizations are left to future work.

Second, we measured the extra time required for performing a top-k test, see Algorithm 10. On average over both benchmarks and all parameters, a top-k test needed 4.3K ns. This time comprises a selectivity estimation lookup for the binding probability and the score probability computation. In contrast, a sorted (random) access took 26.8K ns (1.7M ns) on average. Thus, a top-k test is fairly cheap in comparison to a sorted and random access, respectively.

Effectiveness Results. For inspecting the effectiveness of our score distribution learning, we measured how well the trained distributions fit the observed complete binding scores. More precisely, we applied the well-known Kolmogorov-Smirnov test [68], which measures via a p-value in $[0, 1]$ whether a sample comes from the population of a specific distribution. Table 7-b depicts p-values as averages for both benchmarks.

We observed drastic differences between p-values for SP² and DBPSB. SP² results were very promising and reflect that learned distributions accurately capture the true scores of complete bindings. That is, our distribution learning could achieve high-quality approximations: the p-value was 0.34 for distribution e as well as distribution n , and 0.31 for distribution u . With regard to DBPSB, we could not train good distributions. We measured poor p-values: 0.04 for distribution e , 0.01 given distribution n , and 0.02 for distribution u . We explain this with the few training samples available for DBPSB queries, due to their high selectivity. As discussed above, SP² queries featured much more score samples, see Table 7-a. However, the interesting observation is that the overall approach, A-PBRJ, was hardly affected. In fact, A-PBRJ achieved a high precision $\in [0.73, 0.98]$ for DBPSB. This is because the score probabilities are only relevant for the top- k test, if k complete bindings have been computed (see discussion in Section 5.3.3.3). However, for many DBPSB queries, their cardinality was $\leq k$. Thus, the poor distribution quality had little to no effect.

Overall, we can conclude that the probabilistic component trains score distributions in an efficient and effective manner, if sufficient score samples are available.

5.3.5 Related Work

There is a large body of work on top- k query processing for relational data [95]. Most recently, such approaches have been extended to RDF data and SPARQL queries [2, 115]. In particular, we presented the LD-PBRJ operator for top- k join processing over distributed Web data in Chapter 3.

These works *aim at exact and complete top- k results*. However, for many applications result accuracy and completeness is not important. Instead, result computation time is the key factor. We outlined several examples in Section 5.1. In particular, end-user oriented search over Web data is a prime example for those applications.

To foster an efficient result computation (by trading off result accuracy/completeness), *approximate top- k techniques* have been proposed [15, 16, 120, 151, 157]. Most notably, [157] used score statistics to predict the highest possible complete score of a partial binding. Partial results are discarded, if they are not likely to contribute to a top- k result. Focusing on distributed top- k queries, [120] employed histograms to predict aggregated score values over a space of data sources. Anytime measures for top- k processing have been introduced by [15, 16]. For this, the authors used offline score information, e.g., histograms, to predict complete binding scores at runtime. Last, approximate top- k processing under budgetary has been addressed in [151].

However, all such approximate top- k approaches heavily rely on *score statistics at indexing time*. That is, scores must be known at indexing time for computing

statistics, e.g., histograms. However, offline statistics lead to major drawbacks in a Web setting – as outlined in Problem 1 and Problem 2 in Section 5.3.1.1. In contrast, we propose a lightweight system: *we learn our score distributions in a pay-as-you-go manner at runtime*. In fact, our statistics cause only *minor overhead in terms of space and time*, see Theorem 5 and Theorem 6.

5.3.6 Summary

In this section, we introduced a novel approximate join top-k algorithm, the so-called A-PBRJ, thereby addressing Research Question 4:

🔗 Research Question 4

How to enable approximate top-k query processing for hybrid queries over schemaless Web data?

For this research question, we aimed at two hypotheses by means of our A-PBRJ operator:

☑ Hypothesis 7

Given user/query-dependent ranking functions, we can learn score statistics for approximate top-k query processing by means of Bayesian statistics.

On the one hand, with regard to the above hypothesis, we introduced a score learning procedure in Algorithm 9, which allows to train score distribution via Bayesian statistics at runtime. This way, our approach does not require offline information about ranking functions or score distribution, respectively. In fact, every statistic needed can be learned at runtime from score samples observed during query processing.

We empirically validated the efficiency and effectiveness of this score distribution learning. In particular, we showed that our A-PBRJ system could achieve times savings of up to 65%, while maintaining a high precision/recall.

Additionally, we conducted a theoretical analysis in Section 5.3.3.6, thereby showing the effectiveness of the proposed score distribution learning in Theorem 7 and Theorem 8. Moreover, we gave bounds for the approximation error in Theorem 9.

☑ Hypothesis 8

Bayesian statistics allow for lightweight score statistics.

On the other hand, with regard to Hypothesis 8, we showed space and time complexity bounds in Theorem 5 and Theorem 6, respectively. Further, in our evaluation in Section 5.3.4, we provided empirical evidence that Bayesian statistics allow for an efficient implementation of the probabilistic component \mathcal{PC} .

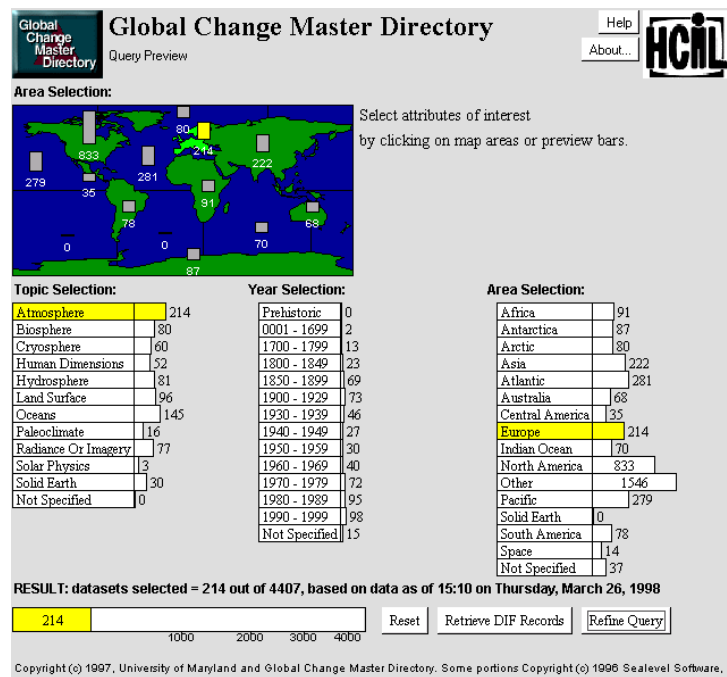


Figure 43: Screenshot of NASA's EOSDIS system.⁴¹

5.4 CONCLUSION

In this chapter, we introduced two new approaches for approximate query processing over Web data. Our first approach resembles a pipeline of operators, which incrementally process hybrid queries in a rank-agnostic manner. In particular, we introduced several novel data synopses and approximate query processing strategies for Web data. In contrast, our second approach is a rank-aware approximate join operator, which discards partial query bindings that probably would not contribute to the final top-k results. For this, we exploit well-known techniques from the field of Bayesian statistics, in order to learn necessary data synopses at runtime. This way, we allow to employ user/query-dependent ranking functions for hybrid queries – a key requirement for an effective search over hybrid Web data.

Both approaches allow a system to handle information needs, which do not require a high result accuracy and completeness, respectively. These information needs frequently occur in end-user oriented systems – as discussed in Section 5.1.

However, driven by the increasing amount of Web data, systems will have a strong need for efficient query processing strategies. Targeting this issue, approximate query processing can help by reporting approximated results early and computing exact/complete results solely on demand. More precisely, such strategies could be seen as a form of *query result preview* [133], which provides initial insights into the query and its potential results, respectively. While a user inspects the result preview, a system can spend additional time for computing refined results. In fact, while inspecting the result preview, a user could refine the query in order to better match her information need [133].

⁴¹<https://www.cs.umd.edu/hcil/eosdis/>, retrieved 2013-10-05.

A prominent example for a system with query result preview functionality is NASA's EOSDIS (Earth Observing System Data and Information System) system [49]. Consider the screenshot in Figure 43. Here, users are presented interactive results previews for NASA's environmental data. Users can refine their queries, e.g., by specifying the specific geographic region of interest.

Generally speaking, we argue that various systems, also those facing information needs that require exact and complete results, may exploit approximate query processing techniques in order to cope with the data size.

CONCLUSION

CONCLUSION

6.1 SUMMARY

In this thesis, we addressed the following research question:

➤ **Overall Research Question**

How to allow for rank-aware and approximate query processing over Web data?

We identified three Web data characteristics, which are crucial for the above research question: schemaless Web data, hybrid Web data, and distributed/low-volume Web data. Based on these Web data characteristics, we split the overall research question into four subquestions, which we addressed in Chapter 3, Chapter 4, and Chapter 5. An overview of these research question, our contributions, and the future/related work is illustrated in Figure 44.

➤ **Research Question 1**

How to enable top-k query processing on highly distributed, schemaless Web data?

In Chapter 3, we aimed at Research Question 1 – targeting an approach for rank-aware join processing over highly distributed and schemaless Web data (Characteristic 1 and Characteristic 3, see Figure 44). For this, we provided Contribution 1.

☞ **Contribution for Research Question 1**

Top-k join processing over Linked Data.

Our top-k join processing approach, LD-PBRJ, allows for a push-based, rank-aware query processing. The push-based nature of our approach is a key difference to state-of-the-art pull-based top-k join processing strategies. This way, Web data sources can be retrieved efficiently in a non-blocking manner.

Moreover, our approach requires only very lightweight statistics for its sorted accesses. Such lightweight statistics enable an easy index maintenance. This is a great advantage with regard to the frequently changing Web data.

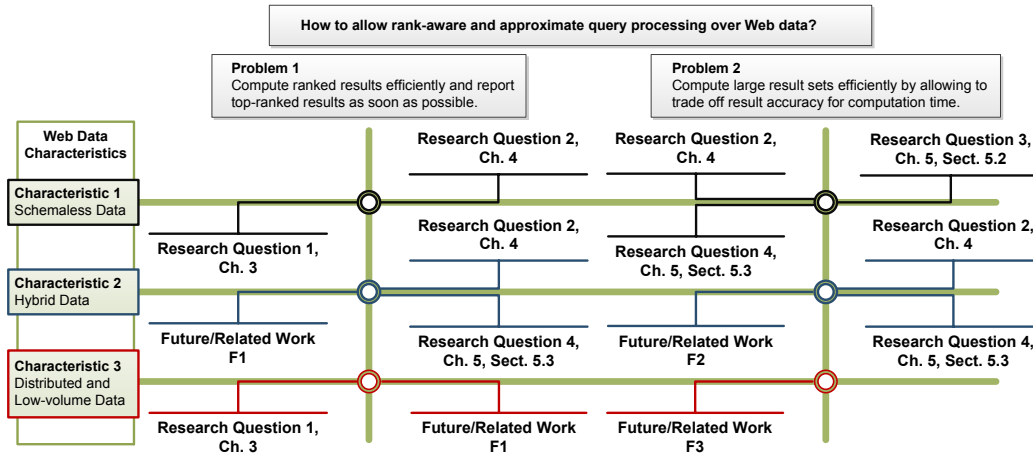


Figure 44: Overview of problems, Web data characteristics, and research questions, which are address in this thesis.

Last, we provided two optimizations for the LD-PBRJ: (1) a tighter bounding strategy and (2) a strategy for the pruning of partial bindings, which can not contribute to the final top-k results. Both optimizations lead to less inputs being materialized and processed. Less inputs, in turn, translate to time savings for the overall query result computation.

Research Question 2

How to allow for efficient and effective selectivity estimates on hybrid, schemaless Web data?

In Chapter 4, we proposed a selectivity estimation approach for the above Research Question 2. More specifically, we provided a tailored solution for selectivity estimation of hybrid queries over text-rich, schemaless Web data (Characteristic 1 and Characteristic 2, see Figure 44). By means of such a selectivity estimation approach, a query optimizer can integrate our rank-aware/approximate join operators into physical query plans.

Contribution for Research Question 2

Selectivity estimation for hybrid and schemaless Web data.

We proposed the BN^+ approach for Research Question 2. Here, we exploited well-known techniques from the field of Bayesian networks to compactly model dependencies in schemaless Web data. In contrast to previous works, our BN^+ approach does not assume a specific data partitioning. Instead, we learn the network structure as well as the network parameters solely from instance data.

Moreover, we integrated string synopses in the BN^+ approach. Via the string synopses, we can efficiently represent large n-gram sample spaces. Thereby, we capture dependencies between structured and unstructured data elements, respectively. Capturing these dependencies in an uniform manner is essential for an effective selectivity estimation of hybrid queries.

📌 Research Question 3

How to enable approximate and incremental query processing on schemaless Web data?

In Chapter 5, we targeted an approach for incremental query processing on schemaless Web data (Characteristic 1 in Figure 44). This way, approximate results can be computed quickly, and can either be reported early or can be refined – based on the given information need.

📌 Contribution for Research Question 3

Approximate and incremental query processing over Web data.

With regard to Research Question 3, we proposed a pipeline comprising four phases: entity search (ES), approximate structure matching (ASM), structure-based result refinement (SRR), and structure-based result computation (SRC). The first three phases compute approximate results, which can be refined (if necessary) by the next phase. Only the very last phase, structure-based result computation (SRC), produces exact results. Throughout the phases, we exploit synopses and join techniques that are well-suited for schemaless Web data.

Moreover, two of our phases – approximate structure matching (ASM) and structure-based result refinement (SRR) – exploit novel approximate join processing techniques. This way, we iteratively and approximately process expensive relation query patterns, respectively.

📌 Research Question 4

How to enable approximate top-k query processing for hybrid queries over schemaless Web data?

Last, we addressed Research Question 4 in Chapter 5. Here, we are concerned with rank-aware approximate query processing. More specifically, instead of computing false positive/negative query results, we compute false positive/negative top-k query results. For this, we exploited a lightweight data synopsis, which is tailored towards hybrid and schemaless Web data (Characteristic 1 and Characteristic 2 in Figure 44).

📌 Contribution for Research Question 4

Approximate top-k query processing for hybrid queries over Web data.

We introduced a novel approximate top-k join algorithm framework, the A-PBRJ, for the above Research Question 4. Within our framework, we rely on Bayesian statistics for training lightweight score statistics during query processing. This way, we allow for user/query-dependent ranking functions, which are crucial for an effective ranking of hybrid queries over text-rich Web data.

Additionally, the lightweight nature of our lightweight score statistics leads to a very low index maintenance overhead. In fact, we showed our approach to have

only a constant space complexity and a runtime complexity that is bounded by the result size. Given the rapidly changing Web data, such a lightweight synopsis and its low index maintenance overhead is a great advantage.

6.2 FUTURE WORK

In the following, we will briefly outline relevant future work with regard to our overall research question. We also depicted such future works in Figure 44.

* Future Work and Related Work – F1

Sorted access for top-k join processing over distributed hybrid Web data.

We proposed a lightweight sorted access implementation for highly distributed Web data in Chapter 3. Intuitively speaking, we presented a source index, which maps triple patterns to potentially matching data sources. For every such data source, we also captured the minimal and maximal ranking score of its triples, respectively.

However, with regard to hybrid Web data, a source index quickly becomes very expensive in terms of space as well as maintenance. Recall that hybrid queries feature a contains semantic. Therefore, a suitable index structure would have to map a pattern $\langle s, a, w \rangle$ to every sources, which comprises a triple that contains attribute a and keyword w .

For this, recent work [104] introduced source index implementations, which are well-suited for hybrid queries and hybrid Web data, respectively. However, such approaches do not incorporate ranking scores. In particular, existing works on index structures for hybrid Web data do not provide sorted accesses. This is a hard problem, because ranking functions for hybrid queries are oftentimes user/query-dependent [12, 36, 156]. Therefore, offline score statistics can not be computed for these ranking functions. Note, for the selection top-k problem, previous works addressed similar problems [33, 92].

Moreover, relying on centralized top-k join query processing over distributed Web data leads to expensive data shipping [103]. Previous work [51] targeted this problem and proposed a distributed top-k join. However, the authors in [51] exploit complex histogram score statistics. Such statistics are very expensive to maintain. Furthermore, these statistics require all scores to be known at indexing time. This strongly restricts the possible ranking functions. In particular, no user/query-dependent ranking function [12, 36, 156] could be used.

* Future Work and Related Work – F2

Index structures for approximate query processing over hybrid Web data.

We presented several index structures for our incremental query processing pipeline in Chapter 5. In particular, we proposed a simple index for the entity search (ES) phase, see Section 5.2.3.2. Recent work [104] presented more sophisticated indexes for hybrid queries and hybrid Web data, respectively. However,

since we do not require exact query results, we could develop index structures that allow for approximate matching of hybrid entity queries. That is, these index structures enable to match a given pattern with a degree of precision. This way, the entity search (ES) phase could evaluate entity queries with increasing degree of precision – as dictated by a given information

Moreover, the entity search (ES) phase could exploit selectivity estimation techniques – as discussed in Chapter 4 – to decide which hybrid entity patterns to evaluate with what degree of precision. Essentially, this could be conceived as a form of query optimization. That is, a query optimizer decides which pattern causes what costs, and should therefore be matched with a particular precision.

Note, above ideas are highly related to approximate query processing techniques, which relax a given user query [86, 88, 89, 90, 136]. However, in contrast to existing works, we argue that the focus should lie on two aspects: (1) Incremental query processing: a system should always be able to compute exact results. (2) Structured query optimization: a system should decide how to relax a query based on query optimization techniques.

* Future Work and Related Work – F3

Distributed approximate/incremental query processing.

Our indexes structures and query processing techniques in Chapter 5 are centralized. That is, we rely on data shipping [103] in order to handle distributed data sources. In particular, with regard to index structures discussed for our incremental query processing pipeline in Section 5.2, future work should aim at more sophisticated query processing techniques. More precisely, a query processing pipeline should take the data source characteristics (e.g., location, latency, access capability) into account. This way, the query processing phases could first exploit “cheap” data sources, e.g., data sources which have a low latency, while using “expensive” data sources only if necessary.

In contrast to existing works on distributed query processing [103], this adds an additional dimension in the optimization problem: the information need. More precisely, a suitable approach needs to estimate (1) the costs of processing a given query by means of specific data sources and (2) the expected query result quality. Thus, the approach should have the freedom to trade off result quality in favor of a cheaper query plan/distribution.

PUBLICATIONS

- [1] Thanh Tran, Günter Ladwig, and Andreas Wagner. Approximate and Incremental Processing of Complex Queries against the Web of Data. In *Proceedings of the Database and Expert Systems Applications Conference (DEXA)*, 2011.
- [2] Andreas Wagner, Thanh Tran Duc, Günter Ladwig, Andreas Harth, and Rudi Studer. Top-k Linked Data query processing. In *Proceedings of the Extended Semantic Web Conference (ESWC)*, 2012.
- [3] Andreas Wagner, Veli Bicer, and Thanh Tran. Selectivity Estimation for Hybrid Queries over Text-Rich Data Graphs. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 2013.
- [4] Andreas Wagner, Veli Bicer, and Thanh Tran. Pay-as-you-go Approximative Top-k Join Processing for the Web of Data. In *Proceedings of the Extended Semantic Web Conference (ESWC)*, 2014.

REFERENCES

- [5] An Implementation of an Efficient Algorithm for Bisimulation Equivalence. *Sci. Comput. Program.*, 13(2-3):219–236, 1990.
- [6] Semantic link based top-K join queries in P2P networks. In Les Carr, David De Roure, Arun Iyengar, Carole A. Goble, and Michael Dahlin, editors, *Proceedings of the International Conference on World Wide Web (WWW)*, pages 1005–1006, 2006.
- [7] SemSearchPro - Using semantics throughout the search process. *Journal of Web Semantics*, 9(4):349–364, 2011.
- [8] *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation. World Wide Web Consortium, 2012. URL <http://www.w3.org/TR/owl2-overview/>.
- [9] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Scalable semantic web data management using vertical partitioning. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 411–422, 2007.
- [10] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. Join synopses for approximate query answering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 275–286, 1999.
- [11] B. Adida, M. Birbeck, S. McCarron, and I. Herman. *RDFa 1.1 Core - Second Edition*. W3C Recommendations. World Wide Web Consortium, 2013. URL <http://www.w3.org/TR/2013/REC-rdfa-core-20130822/>.
- [12] Rakesh Agrawal, Ralf Rantza, and Evimaria Terzi. Context-sensitive ranking. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 383–394, 2006.
- [13] Boanerges Aleman-Meza, Farshad Hakimpour, I. Budak Arpinar, and Amit P. Sheth. SwetoDblp ontology of Computer Science publications. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(3):151–155, 2007.
- [14] Giuseppe Amato, Fausto Rabitti, Pasquale Savino, and Pavel Zezula. Region Proximity in Metric Spaces and Its Use for Approximate Similarity Search. *ACM Trans. Inf. Syst.*, 21(2):192–227, 2003.
- [15] Benjamin Arai, Gautam Das, Dimitrios Gunopulos, and Nick Koudas. Any-time Measures for Top-k Algorithms. In *Proceedings of the International Conference on Very Large Data Bases*, pages 914–925, 2007.

- [16] Benjamin Arai, Gautam Das, Dimitrios Gunopulos, and Nick Koudas. Any-time measures for top-k algorithms on exact and fuzzy data sets. *The VLDB Journal*, 18(2):407–427, 2009.
- [17] Carlos Aranda, Olivier Corby, Souripriya Das, Lee Feigenbaum, Paul Gearon, Birte Glimm, Steve Harris, Sandro Hawke, Ivan Herman, Nicholas Humfrey, Nico Michaelis, Chimezie Ogbuji, Matthew Perry, Alexandre Passant, Axel Polleres, Eric Prud’hommeaux, Andy Seaborne, and Gregory Williams, editors. *SPARQL 1.1 Overview*. W3C Recommendation. World Wide Web Consortium, 2013. URL <http://www.w3.org/TR/sparql11-overview/>.
- [18] Brian Babcock, Surajit Chaudhuri, and Gautam Das. Dynamic Sample Selection for Approximate Query Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 539–550, 2003.
- [19] Akanksha Baid, Ian Rae, Jiexing Li, AnHai Doan, and Jeffrey Naughton. Toward Scalable Keyword Search over Relational Data. *Proc. VLDB Endow.*, 3(1-2):140–149, 2010.
- [20] Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski, and Uwe Thaden. Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 174–185, 2005.
- [21] D. Becket. *RDF 1.1 N-Triples*. W3C Recommendation. World Wide Web Consortium, 2014. URL <http://www.w3.org/TR/n-triples/>.
- [22] J.M. Bernardo and A.F.M. Smith. *Bayesian Theory*. John Wiley & Sons, 2007.
- [23] Patrick Billingsley. *Probability and Measure*. Wiley-Interscience, 1995.
- [24] 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.
- [25] Christian Bizer, Kai Eckert, Robert Meusel, Hannes Mühleisen, Michael Schuhmacher, and Johanna Völker. Deployment of RDFa, Microdata, and Microformats on the Web – A Quantitative Analysis. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2013.
- [26] Paul E. Black and Vreda Pieterse. Levenshtein Distance, 2013. URL <http://www.nist.gov/dads/HTML/Levenshtein.html>.
- [27] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.
- [28] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The Skyline Operator. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 421–430, 2001.

- [29] Dan Brickley and R. V. Guha, editors. *RDF Schema 1.1*. W3C Recommendation. World Wide Web Consortium, 2014. URL <http://www.w3.org/TR/rdf-schema>.
- [30] Peter Buneman, Susan Davidson, Mary Fernandez, and Dan Suciu. Adding Structure to Unstructured Data. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 336–350, 1997.
- [31] Kaushik Chakrabarti, Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Approximate Query Processing Using Wavelets. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 111–122, 2000.
- [32] Kevin Chen-Chuan Chang and Seung-won Hwang. Minimal Probing: Supporting Expensive Predicates for Top-k Queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 346–357, 2002.
- [33] Kevin Chen-Chuan Chang and Seung-won Hwang. Minimal probing: Supporting expensive predicates for top-k queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 346–357, 2002.
- [34] Yuan-Chi Chang, Lawrence Bergman, Vittorio Castelli, Chung-Sheng Li, Ming-Ling Lo, and John R. Smith. The Onion Technique: Indexing for Linear Optimization Queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 391–402, 2000.
- [35] Surajit Chaudhuri, Venkatesh Ganti, and Luis Gravano. Selectivity Estimation for String Predicates: Overcoming the Underestimation Problem. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 227–238, 2004.
- [36] Surajit Chaudhuri, Gautam Das, Vagelis Hristidis, and Gerhard Weikum. Probabilistic information retrieval approach for ranking of database query results. *ACM Trans. Database Syst.*, 31(3):1134–1168, 2006.
- [37] Surajit Chaudhuri, Bee-Chung Chen, Venkatesh Ganti, and Raghav Kaushik. Example-driven design of efficient record matching queries. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 327–338, 2007.
- [38] Qun Chen, Andrew Lim, and Kian Win Ong. D(k)-index: an adaptive structural summary for graph-structured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 134–144, 2003.
- [39] Yan Chen, Jianbo Ou, Yu Jiang, and Xiaofeng Meng. HStar – a Semantic Repository for Large Scale OWL Documents. In *Proceedings of the Asian Conference on The Semantic Web (ASWC)*, pages 415–428, 2006.

- [40] Yen-Yu Chen, Torsten Suel, and Alexander Markowetz. Efficient Query Processing in Geographic Web Search Engines. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 277–288, 2006.
- [41] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.
- [42] Joel Coffman and Alfred C. Weaver. A framework for evaluating database keyword search strategies. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 729–738, 2010.
- [43] Gao Cong, Christian S. Jensen, and Dingming Wu. Efficient Retrieval of the Top-k Most Relevant Spatial Web Objects. *Proc. VLDB Endow.*, 2(1): 337–348, 2009.
- [44] Olivier Corby, Rose Dieng-Kuntz, Catherine Faron-Zucker, and Fabien L. Gandon. Searching the Semantic Web: Approximate Query Processing Based on Ontologies. *IEEE Intelligent Systems*, 21(1):20–27, 2006.
- [45] R. Cyganiak. A relational algebra for SPARQL. Technical report, HP Laboratories Bristol, 2005. URL <http://www.hpl.hp.com/techreports/2005/HPL-2005-170.html>.
- [46] Roberto De Virgilio, Antonio Maccioni, and Riccardo Torlone. A similarity measure for approximate querying over rdf data. In *Proceedings of the Joint EDBT/ICDT Workshops*, pages 205–213, 2013.
- [47] Roberto De Virgilio, Antonio Maccioni, and Riccardo Torlone. Approximate querying of RDF graphs via path alignment. *Distrib. Parallel Databases*, 2014 (to appear).
- [48] Amol Deshpande, Minos N. Garofalakis, and Rajeev Rastogi. Independence is Good: Dependency-Based Histogram Synopses for High-Dimensional Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 199–210, 2001.
- [49] Khoa Doan, Catherine Plaisant, Ben Shneiderman, and Tom Bruns. Query Previews for Networked Information Systems: A Case Study with NASA Environmental Data. *SIGMOD Record*, 26(1):75–81, 1997.
- [50] Donko Donjerkovic and Raghu Ramakrishnan. Probabilistic Optimization of Top N Queries. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 411–422, 1999.
- [51] Christos Doulkeridis, Akrivi Vlachou, Kjetil Nørkvåg, Yannis Kotidis, and Neoklis Polyzotis. Processing of Rank Joins in Highly Distributed Systems. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 606–617, 2012.

- [52] Shady Elbassuoni, Maya Ramanath, Ralf Schenkel, Marcin Sydow, and Gerhard Weikum. Language-model-based ranking for queries on RDF-graphs. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 977–986, 2009.
- [53] Ramez Elmasri and Shamkant Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 2010.
- [54] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 102–113, 2001.
- [55] Jonathan Finger and Neoklis Polyzotis. Robust and efficient algorithms for rank join evaluation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 415–428, 2009.
- [56] Michael Franklin, Alon Halevy, and David Maier. From Databases to Dataspaces: A New Abstraction for Information Management. *SIGMOD Record*, 34(4):27–33, 2005.
- [57] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1300–1309, 1999.
- [58] Minos N. Garofalakis and Phillip B. Gibbon. Approximate Query Processing: Taming the TeraBytes. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, page 725, 2001.
- [59] Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2007.
- [60] Lise Getoor, Benjamin Taskar, and Daphne Koller. Selectivity estimation using probabilistic models. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 461–472, 2001.
- [61] Roy Goldman and Jennifer Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 436–445. Morgan Kaufmann Publishers Inc., 1997.
- [62] Goetz Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2):73–169, 1993.
- [63] Goetz Graefe. The Cascades Framework for Query Optimization. *IEEE Data Eng. Bull.*, 18(3):19–29, 1995.
- [64] Goetz Graefe and William J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 209–218, 1993.

- [65] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Optimizing Multi-Feature Queries for Image Databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 419–428, 2000.
- [66] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Towards Efficient Multi-Feature Queries in Heterogeneous Environments. In *IEEE International Conference on Information Technology: Coding and Computing (ITCC)*, page 622, 2001.
- [67] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2–3):158–182, 2005.
- [68] A. Gut. *Probability: A Graduate Course: A Graduate Course*. Springer, 2012.
- [69] Antonin Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 47–57, 1984.
- [70] L. M. Haas, J. C. Freytag, G. M. Lohman, and H. Pirahesh. Extensible query processing in starburst. *SIGMOD Record*, 18(2):377–388, 1989.
- [71] Alon Halevy, Michael Franklin, and David Maier. Principles of Dataspace Systems. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 1–9, 2006.
- [72] Alon Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.
- [73] Andreas Harth and Stefan Decker. Optimized Index Structures for Querying RDF from the Web. In *Proceedings of the Latin American Web Congress (LA-WEB)*, pages 71–80, 2005.
- [74] Andreas Harth, Sheila Kinsella, and Stefan Decker. Using Naming Authority to Rank Data and Ontologies for Web Search. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 277–292, 2009.
- [75] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. Data summaries for on-demand queries over linked data. In *Proceedings of the International Conference on World Wide Web (WWW)*, pages 411–420, 2010.
- [76] Olaf Hartig. An Overview on Execution Strategies for Linked Data Queries. *Datenbank-Spektrum*, 13(2):89–99.
- [77] Olaf Hartig. Zero-Knowledge Query Planning for an Iterator Implementation of Link Traversal Based Query Execution. In *Proceedings of the 8th Extended Semantic Web Conference on The Semantic Web (ESWC)*, pages 154–169, 2011.
- [78] Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. Executing SPARQL Queries over the Web of Linked Data. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 293–309, 2009.

- [79] Daniel M. Herzig. *Ranking for Web Data Search Using On-The-Fly Data Integration*. PhD thesis, Karlsruhe Institute of Technology, Karlsruhe, 2013.
- [80] Ian Hickson, editor. *HTML Microdata*. W3C Recommendation. World Wide Web Consortium, 2012. URL <http://www.w3.org/TR/microdata/>.
- [81] P. Hitzler, M. Krotzsch, and S. Rudolph. *Foundations of Semantic Web Technologies*. Taylor & Francis, 2011.
- [82] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph, editors. *OWL 2 Web Ontology Language: Primer*. W3C Recommendation. World Wide Web Consortium, 2012. URL <http://www.w3.org/TR/owl2-primer/>.
- [83] P.D. Hoff. *A first course in Bayesian statistical methods*. Springer, 2009.
- [84] Katja Hose, Ralf Schenkel, Martin Theobald, and Gerhard Weikum. Database foundations for scalable RDF processing. In *Proceedings of the International Conference on Reasoning Web (RW)*, pages 202–249, 2011.
- [85] Katja Hose, Ralf Schenkel, Martin Theobald, and Gerhard Weikum. Database Foundations for Scalable RDF Processing. In *Proceedings of the International Conference on Reasoning Web (RW)*, pages 202–249, 2011.
- [86] Hai Huang and Chengfei Liu. Query relaxation for star queries on rdf. In *Proceedings of the International Conference on Web Information Systems Engineering (WISE)*, pages 376–389, 2010.
- [87] Hai Huang and Chengfei Liu. Estimating selectivity for joined RDF triple patterns. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 1435–1444, 2011.
- [88] Hai Huang, Chengfei Liu, and Xiaofang Zhou. Computing Relaxed Answers on RDF Databases. In *Proceedings of the International Conference on Web Information Systems Engineering (WISE)*, pages 163–175, 2008.
- [89] Hai Huang, Chengfei Liu, and Xiaofang Zhou. Approximating Query Answering on RDF Databases. *World Wide Web*, 15(1):89–114, 2012.
- [90] Carlos A. Hurtado, Alexandra Poulouvasilis, and Peter T. Wood. A Relaxed Approach to RDF Querying. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 314–328, 2006.
- [91] Carlos A. Hurtado, Alexandra Poulouvasilis, and Peter T. Wood. Ranking Approximate Answers to Semantic Web Queries. In *Proceedings of the 8th Extended Semantic Web Conference on The Semantic Web (ESWC)*, pages 263–277, 2009.
- [92] Seung-Won Hwang and K.C.-C. Chang. Probe minimization by schedule optimization: Supporting top-k queries with expensive predicates. *IEEE Transactions on Knowledge and Data Engineering*, 19(5):646–662, 2007.

- [93] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. *The VLDB Journal*, 13(3):207–221, 2004.
- [94] Ihab F. Ilyas, Walid G. Aref, Ahmed K. Elmagarmid, Hicham G. Elmongui, Rahul Shah, and Jeffrey Scott Vitter. Adaptive Rank-aware Query Optimization in Relational Databases. *ACM Trans. Database Syst.*, 31(4):1257–1304, 2006.
- [95] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A Survey of Top-k Query Processing Techniques in Relational Database Systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, 2008.
- [96] Yannis E. Ioannidis. Approximations in database systems. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 16–30, 2002.
- [97] Ian Jacobs and Norman Walsh, editors. *Architecture of the World Wide Web, Volume One*. W3C Recommendation. World Wide Web Consortium, 2004. URL <http://www.w3.org/TR/webarch/>.
- [98] H. V. Jagadish, Olga Kapitskaia, Raymond T. Ng, and Divesh Srivastava. One-dimensional and multi-dimensional substring selectivity estimation. *The VLDB Journal*, 9(3):214–230, 2000.
- [99] Liang Jin and Chen Li. Selectivity Estimation for Fuzzy String Predicates in Large Data Sets. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 397–408, 2005.
- [100] Raghav Kaushik, Philip Bohannon, Jeffrey F. Naughton, and Henry F. Korth. Covering indexes for branching path queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 133–144, 2002.
- [101] Kifer, Michael and Lausen, Georg. F-logic: A Higher-order Language for Reasoning About Objects, Inheritance, and Scheme. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 134–146, 1989.
- [102] D. Koller and N. Friedman. *Probabilistic Graphical Models*. The MIT Press, 2009.
- [103] Donald Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4), 2000.
- [104] Günter Ladwig. *Efficient Optimization and Processing of Queries over Text-rich Graph-structured Data*. PhD thesis, Karlsruhe Institute of Technology, 2013.
- [105] Günter Ladwig and Thanh Tran. Linked Data Query Processing Strategies. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2010.
- [106] Günter Ladwig and Thanh Tran. SIHJoin: Querying Remote and Local Linked Data. In *Proceedings of the 8th Extended Semantic Web Conference on The Semantic Web (ESWC)*, pages 139–153, 2011.

- [107] Hongrae Lee, Raymond T. Ng, and Kyuseok Shim. Extending Q-Grams to Estimate Selectivity of String Matching with Low Edit Distance. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 195–206, 2007.
- [108] Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. Static analysis and optimization of semantic web queries. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 89–100, 2012.
- [109] Chengkai Li, Kevin Chen-Chuan Chang, Ihab F. Ilyas, and Sumin Song. RankSQL: Query Algebra and Optimization for Relational Top-k Queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 131–142, 2005.
- [110] Chengkai Li, Kevin Chen-Chuan Chang, and Ihab F. Ilyas. Supporting Ad-hoc Ranking Aggregates. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 61–72, 2006.
- [111] Feifei Li, Ke Yi, and Wangchao Le. Top-k Queries on Temporal Data. *The VLDB Journal*, 19(5):715–733, 2010.
- [112] Ling Liu and M. Tamer Özsu, editors. *Encyclopedia of Database Systems*. Springer, 2009.
- [113] Yi Luo, Xuemin Lin, Wei Wang, and Xiaofang Zhou. Spark: Top-k Keyword Query in Relational Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 115–126, 2007.
- [114] Yi Luo, Wei Wang, Xuemin Lin, Xiaofang Zhou, Jianmin Wang, and Keqiu Li. SPARK2: Top-k Keyword Query in Relational Databases. *IEEE Transactions on Knowledge and Data Engineering*, 23(12):1763–1780, 2011.
- [115] Sara Magliacane, Alessandro Bozzon, and Emanuele Della Valle. Efficient execution of top-k SPARQL queries. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 344–360, 2012.
- [116] Nikos Mamoulis, Man Lung Yiu, Kit Hung Cheng, and David W. Cheung. Efficient top-k aggregation of ranked inputs. *ACM Trans. Database Syst.*, 32(3), 2007.
- [117] D. Martinenghi and M. Tagliasacchi. Cost-Aware Rank Join with Random and Sorted Access. *IEEE Transactions on Knowledge and Data Engineering*, 24(12):2143–2155, 2012.
- [118] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 448–459, 1998.
- [119] M. Meila and M. I. Jordan. Learning with mixtures of trees. *The Journal of Machine Learning Research*, 1:1–48, 2001.

- [120] Sebastian Michel, Peter Triantafillou, and Gerhard Weikum. KLEE: a framework for distributed top-k query algorithms. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 637–648, 2005.
- [121] Peter Mika, Edgar Meij, and Hugo Zaragoza. Investigating the Semantic Gap through Query Log Analysis. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 441–455, 2009.
- [122] Jacob Paul Morgenstein. *Computer Based Management Information Systems Embodying Answer Accuracy As a User Parameter*. PhD thesis, 1981.
- [123] Morsey, Mohamed and Lehmann, Jens and Auer, Sören and Ngomo, Axel-Cyrille Ngonga. DBpedia SPARQL Benchmark: Performance Assessment with Real Queries on Real Data. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 454–469, 2011.
- [124] M. Muralikrishna. Improved Unnesting Algorithms for Join Aggregate SQL Queries. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 91–102, 1992.
- [125] Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. Supporting Incremental Join Queries on Ranked Inputs. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 281–290, 2001.
- [126] Wolfgang Nejdl, Wolf Siberski, Uwe Thaden, and Wolf-Tilo Balke. Top-k Query Evaluation for Schema-Based Peer-to-Peer Networks. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 137–151. 2004.
- [127] T. Neumann and G. Moerkotte. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 984–994, 2011.
- [128] Thomas Neumann. *Efficient generation and execution of DAG-structured query graphs*. PhD thesis, University of Mannheim, 2005.
- [129] Thomas Neumann and Gerhard Weikum. RDF-3X: a RISC-style engine for RDF. *PVLDB*, 1(1):647–659, 2008.
- [130] Thomas Neumann and Gerhard Weikum. Scalable join processing on very large RDF graphs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 627–640, 2009.
- [131] Thomas Neumann, Matthias Bender, Sebastian Michel, Ralf Schenkel, Peter Triantafillou, and Gerhard Weikum. Distributed Top-k Aggregation Queries at Large. *Distrib. Parallel Databases*, 26(1):3–27, 2009.
- [132] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, 1999. URL <http://ilpubs.stanford.edu:8090/422/>.

- [133] Catherine Plaisant, Ben Shneiderman, Khoa Doan, and Tom Bruns. Interface and Data Architecture for Query Preview in Networked Information Systems. *ACM Trans. Inf. Syst.*, 17(3):320–341, 1999.
- [134] Neoklis Polyzotis, Minos Garofalakis, and Yannis Ioannidis. Approximate XML query answers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 263–274, 2004.
- [135] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, (2):294–305, 1996.
- [136] Alexandra Poulouvasilis and Peter T. Wood. Combining approximation and relaxation in semantic web path queries. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 631–646, 2010.
- [137] Christopher Re, Nilesh N. Dalvi, and Dan Suciu. Efficient Top-k Query Evaluation on Probabilistic Data. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 886–895, 2007.
- [138] Guy Sagy, Izchak Sharfman, Daniel Keren, and Assaf Schuster. Top-k Vectorial Aggregation Queries in a Distributed Environment. *J. Parallel Distrib. Comput.*, 71(2):302–315, 2011.
- [139] Sherif Sakr and Ghazi Al-Naymat. Relational processing of RDF queries: a survey. *SIGMOD Record*, 38(4):23–28, 2010.
- [140] Florian Schmedding. Incremental SPARQL Evaluation for Query Answering on Linked Data. In *Proceedings of the International Workshop on Consuming Linked Data (COLD)*, 2011.
- [141] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. SP²Bench: A SPARQL Performance Benchmark. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 222–233, 2009.
- [142] Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of SPARQL query optimization. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 4–33, 2010.
- [143] Karl Schnaitter and Neoklis Polyzotis. Evaluating rank joins with optimal cost. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 43–52, 2008.
- [144] Karl Schnaitter and Neoklis Polyzotis. Optimal algorithms for evaluating rank joins in database systems. *ACM Trans. Database Syst.*, 35(1):6–1, 2010.
- [145] Karl Schnaitter, Joshua Spiegel, and Neoklis Polyzotis. Depth estimation for ranking query optimization. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 902–913, 2007.
- [146] Karl Schnaitter, Joshua Spiegel, and Neoklis Polyzotis. Depth Estimation for Ranking Query Optimization. *The VLDB Journal*, 18(2):521–542, 2009.

- [147] Guus Schreiber and Yves Raimond, editors. *RDF 1.1 Primer*. W3C Recommendation. World Wide Web Consortium, 2014. URL <http://www.w3.org/TR/rdf11-primer/>.
- [148] Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-Wesley, 2011.
- [149] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 23–34, 1979.
- [150] Warren Shen, AnHai Doan, Jeffrey F. Naughton, and Raghu Ramakrishnan. Declarative Information Extraction Using Datalog with Embedded Extraction Predicates. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1033–1044, 2007.
- [151] Michal Shmueli-Scheuer, Chen Li, Yosi Mass, Haggai Roitman, Ralf Schenkel, and Gerhard Weikum. Best-Effort Top-k Query Processing Under Budgetary Constraints. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 928–939, 2009.
- [152] M.A. Soliman, I.F. Ilyas, and K.C.-C. Chang. Top-k Query Processing in Uncertain Databases. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 896–905, 2007.
- [153] Steve Speicher, John Arwe, and Ashok Malhotra, editors. *Linked Data Platform 1.0*. W3C Working Draft. World Wide Web Consortium, 2013. URL <http://www.w3.org/TR/ldp/>.
- [154] Joshua Spiegel and Neoklis Polyzotis. Graph-based synopses for relational selectivity estimation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 205–216, 2006.
- [155] Ben Taskar, Eran Segal, and Daphne Koller. Probabilistic classification and clustering in relational data. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 870–876, 2001.
- [156] Aditya Telang, Chengkai Li, and Sharma Chakravarthy. One Size Does Not Fit All: Toward User- and Query-Dependent Ranking for Web Databases. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1671–1685, 2012.
- [157] Martin Theobald, Gerhard Weikum, and Ralf Schenkel. Top-k query evaluation with probabilistic guarantees. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 648–659, 2004.
- [158] Duc Thanh Tran. *Semantic Web Search - A Process-Oriented Perspective on Data Retrieval on the Semantic Web*. PhD thesis, Karlsruhe Institute of Technology, Karlsruhe, 2010.
- [159] Duc Thanh Tran and Günter Ladwig. Structure Index for RDF Data. In *Workshop on Semantic Data Management*, 2010.

- [160] Duc Thanh Tran and Peter Mika. Semantic Search - Systems, Concepts, Methods and the Communities behind It, 2012. URL <http://sites.google.com/site/kimducthanh/publication/semsearch-survey.pdf>. Under Submission.
- [161] Petros Tsialiamanis, Lefteris Sidiropoulos, Irini Fundulaki, Vassilis Christophides, and Peter Boncz. Heuristics-based query optimisation for SPARQL. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 324–335, 2012.
- [162] Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. Lightweight Graphical Models for Selectivity Estimation Without Independence Assumptions. *PVLDB*, 4(11):852–863, 2011.
- [163] Akrivi Vlachou, Christos Doukeridis, and Kjetil Nørnvåg. Distributed Top-k Query Processing by Exploiting Skyline Summaries. *Distrib. Parallel Databases*, 30(3-4):239–271, 2012.
- [164] Daisy Zhe Wang, Long Wei, Yunyao Li, Frederick Reiss, and Shivakumar Vaithyanathan. Selectivity estimation for extraction operators over text data. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 685–696, 2011.
- [165] Gerhard Weikum. DB&IR: both sides now. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 25–30, New York, NY, USA, 2007.
- [166] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. Hexastore: sextuple indexing for semantic web data management. *PVLDB*, 1(1):1008–1019, 2008.
- [167] Eric W. Weisstein. Graph Eccentricity. From MathWorld – A Wolfram Web Resource. URL <http://mathworld.wolfram.com/GraphEccentricity.html>.
- [168] Kevin Wilkinson, Craig Sayers, Harumi A. Kuno, and Dave Reynolds. Efficient RDF Storage and Retrieval in Jena2. In *Proceedings of the International Workshop on Semantic Web and Databases*, pages 131–150, 2003.
- [169] A.N. Wilschut and P. M G Apers. Dataflow query execution in a parallel main-memory environment. In *Proceedings of the International Conference on Parallel and Distributed Information Systems (PDIS)*, pages 68–77, 1991.
- [170] Minji Wu, Laure Berti-Equille, Amelie Marian, Cecilia M. Procopiuc, and Divesh Srivastava. Processing top-k join queries. *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 860–870, 2010.
- [171] Jeffrey Xu Yu, Lu Qin, and Lijun Chang. Keyword Search in Relational Databases: A Survey. *IEEE Data Eng. Bull.*, 33(1):67–78, 2010.
- [172] Yidong Yuan, Xuemin Lin, Qing Liu, Wei Wang, Jeffrey Xu Yu, and Qing Zhang. Efficient Computation of the Skyline Cube. In *Proceedings of the*

- International Conference on Very Large Data Bases (VLDB)*, pages 241–252, 2005.
- [173] Lei Zhang, Qiaoling Liu, Jie Zhang, Haofen Wang, Yue Pan, and Yong Yu. Semplore: An IR Approach to Scalable Hybrid Query of Semantic Web Data. In *Proceedings of the International The Semantic Web and Asian Conference on Asian Semantic Web Conference (ISWC/ASWC)*, pages 652–665, 2007.
- [174] Zhen Zhang, Seung-won Hwang, Kevin Chen-Chuan Chang, Min Wang, Christian A. Lang, and Yuan-chi Chang. Boolean + Ranking: Querying a Database by K-constrained Optimization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 359–370, 2006.
- [175] Keping Zhao, Shuigeng Zhou, K.-L. Tan, and Aoying Zhou. Supporting ranked join in peer-to-peer networks. In *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA)*, pages 796–800, 2005.
- [176] Keping Zhao, Shuigeng Zhou, and Aoying Zhou. Towards Efficient Ranked Query Processing in Peer-to-Peer Networks. In *Proceedings of the Joint Chinese-German Workshop, Cognitive Systems*, pages 145–160. 2007.

LIST OF FIGURES

Figure 1	HTML IMDB Web page about the movie “Roman Holiday”.	3
Figure 2	HTML source code for the “Roman Holiday” Web page.	4
Figure 3	Linked MDB page about “Roman Holiday”.	5
Figure 4	The semantic search process.	6
Figure 5	Search for movies with keyword “holiday”.	8
Figure 6	Problems and research questions in this thesis.	12
Figure 7	RDF graph for the running example.	18
Figure 8	Hybrid query graph for the running example.	19
Figure 9	Query processing overview.	24
Figure 10	Query processing example.	25
Figure 11	Scan operator and equi-join operator.	26
Figure 12	Abstract iterator operator.	29
Figure 13	Overview of top-k strategies.	30
Figure 14	Top-k processing example.	30
Figure 15	Classification of the LD-PBRJ approach.	38
Figure 16	Data graph for the running example in Chapter 3.	39
Figure 17	Listing of Linked Data sources.	41
Figure 18	Query graph for the running example in Chapter 3.	42
Figure 19	Push-based query plan.	44
Figure 20	Rank-aware query plan for the running example.	48
Figure 21	Detailed view on the LD-PBRJ operator.	53
Figure 22	Overview of the evaluation results for Chapter 3.	61
Figure 23	Detailed evaluation results for Chapter 3.	62
Figure 24	Context of the selectivity estimation in Chapter 5.	69
Figure 25	Bayesian Network for the running example in Chapter 4.	74
Figure 26	Evaluation results for Chapter 4.	96
Figure 27	Classification of the A-PBRJ approach.	103
Figure 28	Overview of Chapter 5.	104
Figure 29	Generic approximate query processing approach.	105
Figure 30	Query processing pipeline for Section 5.2.	110
Figure 31	Data/query graph for the running example in Section 5.2.	111
Figure 32	Data synopses for the query processing pipeline.	114
Figure 33	Entity neighborhoods for the running example.	116
Figure 34	Evaluation for Section 5.2: computation times.	124
Figure 35	Evaluation for Section 5.2: computation times vs. query shapes.	125
Figure 36	Evaluation for Section 5.2: neighborhood distance vs. computation time.	126
Figure 37	Evaluation for Section 5.2: neighborhood distance vs. precision.	126
Figure 38	Evaluation for Section 5.2: Result precision vs. computation time.	127

Figure 39	Data/query graph for the running example in Section 5.3	132
Figure 40	Approximate rank join tree for the running example. . . .	136
Figure 41	Evaluation for Section 5.3: #inputs vs. threshold and time vs. threshold.	154
Figure 42	Evaluation for Section 5.3: macro-precision vs. threshold and score error vs. threshold.	156
Figure 43	Screenshot of NASA's EOSDIS system.	161
Figure 44	Problems and research questions in this thesis.	165

LIST OF TABLES

Table 1	Dataset statistics for the evaluation in Chapter 4.	93
Table 2	Query statistics for the evaluation in Chapter 4.	93
Table 3	Approximated results for the running example.	113
Table 4	Dataset statistics for the evaluation in Section 5.2.	123
Table 5	Predictive score distributions for the running example. . .	143
Table 6	Query statistics for the evaluation in Section 5.2.	153
Table 7	Evaluation for Section 5.3: score distribution learning. . . .	158

LIST OF ALGORITHMS

1	Pull/Bound Rank Join algorithm framework.	35
2	LD-PBRJ push method.	50
3	LD-PBRJ execute method.	52
4	LD-PBRJ activate method.	52
5	BN ⁺ data synopsis construction algorithm.	87
6	Approximate structure matching (ASM) algorithm.	118
7	Structure-based result refinement (SRR) algorithm.	120
8	A-PBRJ algorithm framework.	139
9	A-PBRJ training algorithm.	145
10	A-PBRJ top-k test algorithm.	147

ACRONYMS

WWW	World Wide Web
RDF	Resource Description Format
RDFS	Resource Description Framework Schema
OWL	Web Ontology Language
SPARQL	SPARQL protocol and RDF query language
HTML	HyperText Markup Language
W3C	World Wide Web Consortium
AQP	approximate query processing
BGP	basic graph pattern
PBRJ	Pull Bound Rank Join
HRJN	hash rank-join
NRJN	nested-loops rank-join
HTTP	Hypertext Transfer Protocol
URI	Uniform Resource Identifier
PRM	probabilistic relational model
BN	Bayesian network
CPD	conditional probability distribution
BFS	breadth-first search
DFS	depth-first search
A-PBRJ	approximate Pull/Bound Rank Join
CLT	Central Limit Theorem
DB	database
IR	Information Retrieval

A

APPENDIX: EVALUATION QUERIES

A.1 EVALUATION QUERIES FOR CHAPTER 3

Queries employed in Section 3.4 – partially based on FedBench benchmark.⁴²

Listing 7: Query 1.

```
1
2 PREFIX dbpedia: <http://dbpedia.org/resource/Category:>
3 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
4 PREFIX owl: <http://www.w3.org/2002/07/owl#>
5 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
6 PREFIX dcterms: <http://purl.org/dc/terms/>
7
8 SELECT * WHERE {
9   ?x dcterms:subject dbpedia:Liberal_democracies .
10  ?x rdfs:label ?l .
11  ?x owl:sameAs ?x2 .
12  ?x2 foaf:name ?n .
13 }
```

Listing 8: Query 2.

```
1
2 PREFIX dbpedia: <http://dbpedia.org/resource/>
3 PREFIX dbowl: <http://dbpedia.org/ontology/>
4 PREFIX owl: <http://www.w3.org/2002/07/owl#>
5 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
6
7 SELECT * WHERE {
8   ?p dbowl:stateOfOrigin dbpedia:Italy .
9   ?p a foaf:Person .
10  ?p owl:sameAs ?p2 .
11 }
```

Listing 9: Query 3.

```
1
2 PREFIX drugbank: <http://www4.wiwiwiss.fu-berlin.de/drugbank/
3   resource/drugbank/>
4 PREFIX owl: <http://www.w3.org/2002/07/owl#>
5
6 SELECT * WHERE {
7   ?d owl:sameAs ?d2 .
8   ?d drugbank:drugCategory ?c .
9   ?d drugbank:casRegistryNumber ?id .
10 }
```

⁴²<http://fedbench.fluidops.net>, retrieved 2013-12-07.

Listing 10: Query 4.

```
1
2 PREFIX dbpedia: <http://dbpedia.org/resource/Category:>
3 PREFIX owl: <http://www.w3.org/2002/07/owl#>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 PREFIX dcterms: <http://purl.org/dc/terms/>
6
7 SELECT * WHERE {
8   ?x dcterms:subject dbpedia:Western_Europe .
9   ?x owl:sameAs ?x2 .
10  ?x2 rdfs:label ?l .
11 }
```

Listing 11: Query 5.

```
1
2 PREFIX dbpedia: <http://dbpedia.org/resource/Category:>
3 PREFIX owl: <http://www.w3.org/2002/07/owl#>
4
5 SELECT * WHERE {
6   ?x dcterms:subject dbpedia:Chancellors_of_Germany .
7   ?x2 owl:sameAs ?x .
8 }
```

Listing 12: Query 6.

```
1
2 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3 PREFIX owl: <http://www.w3.org/2002/07/owl#>
4 PREFIX dcterms: <http://purl.org/dc/terms/>
5 PREFIX mdb: <http://data.linkedmdb.org/resource/director/>
6
7 SELECT * WHERE {
8   mdb:8477 foaf:made ?f .
9   ?f dcterms:date ?d .
10  ?f foaf:page ?p .
11  ?f owl:sameAs ?f2 .
12 }
```

Listing 13: Query 7.

```

1
2 PREFIX owl: <http://www.w3.org/2002/07/owl#>
3 PREFIX dailymed: <http://www4.wiwiss.fu-berlin.de/dailymed/
  resource/dailymed/>
4 PREFIX dailymed_orga: <http://www4.wiwiss.fu-berlin.de/dailymed/
  resource/organization/>
5 PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/
  resource/drugbank/>
6
7 SELECT * WHERE {
8   dailymed_orga:Mylan_Pharmaceuticals_Inc dailymed:producesDrug
9     ?d .
10  ?d dailymed:genericDrug ?gd .
11  ?gd drugbank:possibleDiseaseTarget ?dt .
12  ?dt owl:sameAs ?dt2 .
13 }

```

Listing 14: Query 8.

```

1
2 PREFIX dbpedia: <http://dbpedia.org/resource/>
3 PREFIX dbowl: <http://dbpedia.org/ontology/>
4 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
5 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
6
7 SELECT * WHERE {
8   ?b dbowl:artist dbpedia:The_Beatles .
9   ?b rdfs:label ?l1 .
10  ?b rdfs:label ?l2 .
11  ?b dbowl:previousWork ?a .
12  ?a foaf:depiction ?img .
13 }

```

Listing 15: Query 9.

```

1
2 PREFIX dbowl: <http://dbpedia.org/ontology/>
3 PREFIX owl: <http://www.w3.org/2002/07/owl#>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 PREFIX factbook: <http://www4.wiwiss.fu-berlin.de/factbook/ns#>
6
7 SELECT * WHERE {
8   ?c a dbowl:Country .
9   ?c rdfs:label ?l .
10  ?c owl:sameAs ?c2 .
11  ?c2 factbook:unemploymentrate ?u .
12  ?c2 factbook:literacy_totalpopulation ?p .
13 }

```


Listing 16: Query 10.

```
1
2 PREFIX dbpedia: <http://dbpedia.org/resource/>
3 PREFIX dbowl: <http://dbpedia.org/ontology/>
4 PREFIX owl: <http://www.w3.org/2002/07/owl#>
5 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
6 PREFIX mdb_movie: <http://data.linkedmdb.org/resource/movie>
7
8 SELECT * WHERE {
9   ?f mdb_movie:actor ?a .
10  ?f mdb_movie:featured_film_location ?lc .
11  ?lc rdfs:label ?l .
12  ?f owl:sameAs ?f2 .
13  ?f2 dbowl:music dbpedia:John_Williams .
14 }
```

Listing 17: Query 11.

```
1
2 PREFIX geo-ont: <http://www.geonames.org/ontology#>
3
4 SELECT * WHERE {
5   ?c geo-ont:parentFeature <http://sws.geonames.org/6269131/> .
6   ?c geo-ont:officialName "Cornwall" .
7   ?c geo-ont:nearby ?lc .
8   ?lc geo-ont:name ?n .
9   ?lc a ?t .
10 }
```

Listing 18: Query 12.

```
1
2 PREFIX dbpedia: <http://dbpedia.org/resource/>
3 PREFIX dbprop: <http://dbpedia.org/property/>
4 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
5 PREFIX owl: <http://www.w3.org/2002/07/owl#>
6
7 SELECT * WHERE {
8   ?x dbprop:country dbpedia:Germany .
9   ?x owl:sameAs ?x2 .
10  ?x2 foaf:depiction ?img .
11 }
```

Listing 19: Query 13.

```

1 # FedBench – LD query 1
2
3 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4 PREFIX swc: <http://data.semanticweb.org/ns/swc/ontology#>
5 PREFIX swrc: <http://swrc.ontoware.org/ontology#>
6 PREFIX iswc_2008: <http://data.semanticweb.org/conference/iswc
   /2008/>
7
8 SELECT * WHERE {
9   ?p swc:isPartOf iswc_2008:poster_demo_proceedings .
10  ?p swrc:author ?a .
11  ?a rdfs:label ?l .
12 }

```

Listing 20: Query 14.

```

1 # FedBench – LD query 2
2
3 PREFIX swc: <http://data.semanticweb.org/ns/swc/ontology#>
4 PREFIX swrc: <http://swrc.ontoware.org/ontology#>
5 PREFIX eswc: <http://data.semanticweb.org/conference/eswc/>
6
7 SELECT * WHERE {
8   ?pro swc:relatedToEvent eswc:2010 .
9   ?p swc:isPartOf ?pro .
10  ?p swrc:author ?a .
11 }

```

Listing 21: Query 15.

```

1 # FedBench – LD query 3
2
3 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4 PREFIX swc: <http://data.semanticweb.org/ns/swc/ontology#>
5 PREFIX swrc: <http://swrc.ontoware.org/ontology#>
6 PREFIX iswc_2008: <http://data.semanticweb.org/conference/iswc
   /2008/>
7
8 SELECT * WHERE {
9   ?p swc:isPartOf iswc_2008:poster_demo_proceedings .
10  ?p swrc:author ?a .
11  ?a owl:sameAs ?a2 .
12  ?a rdfs:label ?l .
13 }

```

Listing 22: Query 16.

```
1 # FedBench – LD query 4
2
3 PREFIX swc: <http://data.semanticweb.org/ns/swc/ontology#>
4 PREFIX swrc: <http://swrc.ontoware.org/ontology#>
5 PREFIX eswc: <http://data.semanticweb.org/conference/eswc/>
6
7 SELECT * WHERE {
8   ?r swc:isRoleAt eswc:2010 .
9   ?r swc:heldBy ?x .
10  ?p swrc:author ?a .
11  ?p swc:isPartOf ?pro .
12  ?pro swc:relatedToEvent eswc:2010 .
13 }
```

Listing 23: Query 17.

```
1 # FedBench – LD query 5
2
3 PREFIX dbpedia: <http://dbpedia.org/resource/>
4 PREFIX dbowl: <http://dbpedia.org/ontology/>
5 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
6
7 SELECT * WHERE {
8   ?a dbowl:artist dbpedia:Michael_Jackson .
9   ?a a dbowl:Album .
10  ?a foaf:name ?n .
11 }
```

Listing 24: Query 18.

```
1 # FedBench – LD query 7
2
3 PREFIX gn: <http://www.geonames.org/ontology#>
4
5 SELECT * WHERE {
6   ?x gn:parentFeature <http://sws.geonames.org/2921044/> .
7   ?x gn:name ?n .
8 }
```

Listing 25: Query 19.

```

1 # FedBench – LD query 8
2
3 PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/
  resource/drugbank/>
4 PREFIX owl: <http://www.w3.org/2002/07/owl#>
5 PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
6 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
7
8 SELECT * WHERE {
9   ?drug drugbank:drugCategory
10    <http://www4.wiwiss.fu-berlin.de/drugbank/resource/
  drugcategory/micronutrient> .
11   ?drug drugbank:casRegistryNumber ?id .
12   ?drug owl:sameAs ?s .
13   ?s foaf:name ?o .
14   ?s skos:subject ?sub .
15 }

```

Listing 26: Query 20.

```

1 # FedBench – LD query 10
2
3 PREFIX owl: <http://www.w3.org/2002/07/owl#>
4 PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
5 PREFIX nyt: <http://data.nytimes.com/elements/>
6 PREFIX dbpedia: <http://dbpedia.org/resource/Category:>
7
8 SELECT * WHERE {
9   ?c skos:subject dbpedia:Chancellors_of_Germany .
10   ?c owl:sameAs ?c2 .
11   ?c2 nyt:latest_use ?u .
12 }

```

A.2 EVALUATION QUERIES FOR CHAPTER 4

Below, we present the query load used during our experiments in Chapter 4. Queries for the DBLP dataset are based on [114], while IMDB queries are taken from [42]. All queries are given in RDF NTriples [21] notation.

Listing 27: Queries from DBLP benchmark [114].

```
1
2
3 # @PREFIX dc:
4 # <http://purl.org/dc/elements/1.1/> .
5 # @PREFIX foaf:
6 # <http://xmlns.com/foaf/0.1/> .
7 # @PREFIX rdf:
8 # <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
9 # @PREFIX rdfs:
10 # <http://www.w3.org/2000/01/rdf-schema#> .
11 # @PREFIX dblp:
12 # <http://lsdis.cs.uga.edu/projects/semdis/opus#> .
13
14 # q1
15 ?x rdfs:label "clique" .
16 ?x dblp:last_modified_date "2002-12-09" .
17 ?x rdf:type dblp:Article_in_Proceedings .
18 ?x dblp:author ?y .
19 ?y rdf:type foaf:Person .
20 ?y foaf:name "nikos" .
21
22 # q2
23 ?y rdf:type foaf:Person .
24 ?y foaf:name "nikos" .
25 ?y foaf:name "zotos" .
26
27 # q3
28 ?x rdfs:label "constraint" .
29 ?x dblp:last_modified_date "2005-02-25" .
30 ?x rdf:type dblp:Article_in_Proceedings .
31 ?x dblp:author ?y .
32 ?y rdf:type foaf:Person .
33 ?y foaf:name "chuang" .
34
35 # q4
36 ?x rdfs:label "mining" .
37 ?x rdfs:label "clustering" .
38 ?x dblp:year "2005" .
39 ?x rdf:type dblp:Article .
40 ?x dblp:author ?y .
41 ?y rdf:type foaf:Person .
42 ?y foaf:name "nikos" .
43
44 # q5
```

```
45 ?x rdfs:label "spatial" .
46 ?x dblp:last_modified_date "2006-03-31" .
47 ?x rdf:type dblp:Article_in_Proceedings .
48 ?x dblp:author ?y .
49 ?y rdf:type foaf:Person .
50 ?y foaf:name "patel" .
51
52 # q6
53 ?x rdf:type dblp:Article .
54 ?x rdfs:label "middleware" .
55 ?x dblp:author ?y .
56 ?y rdf:type foaf:Person .
57 ?y foaf:name "zhang" .
58
59 # q7
60 ?x rdf:type dblp:Article_in_Proceedings .
61 ?x rdfs:label "middleware" .
62 ?x rdfs:label "optimal" .
63 ?x dblp:author ?y .
64 ?y rdf:type foaf:Person .
65 ?y foaf:name "ronald" .
66
67 # q8
68 ?x rdf:type dblp:Article_in_Proceedings .
69 ?x rdfs:label "partition" .
70 ?x rdfs:label "relational" .
71 ?x rdfs:label "query" .
72
73 # q9
74 ?x rdf:type dblp:Article_in_Proceedings .
75 ?x rdfs:label "partition" .
76 ?x dblp:author ?y .
77 ?y rdf:type foaf:Person .
78 ?y foaf:name "patel" .
79
80 # q10
81 ?x rdf:type dblp:Proceedings .
82 ?x rdfs:label "recognition" .
83 ?x rdfs:label "speech" .
84 ?x rdfs:label "software" .
85 ?x dc:publisher ?p .
86
87 # q11
88 ?x rdf:type dblp:Proceedings .
89 ?x rdfs:label "data" .
90 ?x rdfs:label "mining" .
91 ?x dc:publisher <http://www.springer.de/> .
92
93 # q12
94 ?x rdf:type dblp:Proceedings .
95 ?x rdfs:label "australia" .
96 ?x rdfs:label "stream" .
97 ?x dc:publisher <http://www.springer.de/> .
```

```
98
99 # q13
100 ?x dblp:year "2002" .
101 ?x rdf:type dblp:Proceedings .
102 ?x rdfs:label "industrial" .
103 ?x rdfs:label "database" .
104 ?x dc:publisher ?p .
105
106 # q14
107 ?x rdf:type dblp:Article_in_Proceedings .
108 ?x dblp:last_modified_date "2006-03-09" .
109 ?x dblp:author ?y .
110 ?y rdf:type foaf:Person .
111 ?y foaf:name "jignesh" .
112
113 # q15
114 ?x rdf:type dblp:Article_in_Proceedings .
115 ?x rdfs:label "algorithm" .
116 ?x rdfs:label "incomplete" .
117 ?x rdfs:label "search" .
118
119 # q16
120 ?x dblp:journal_name "SIGMOD" .
121 ?x rdf:type dblp:Article .
122 ?x rdfs:label "web" .
123 ?x rdfs:label "search" .
124
125 # q17
126 ?x rdf:type dblp:Article_in_Proceedings .
127 ?x rdfs:label "semistructured" .
128 ?x rdfs:label "search" .
129 ?x dblp:author ?y .
130 ?y rdf:type foaf:Person .
131 ?y foaf:name "goldman" .
132
133 # q18
134 ?x rdf:type dblp:Article_in_Proceedings .
135 ?x rdfs:label "query" .
136 ?x rdfs:label "cost" .
137 ?x rdfs:label "optimization" .
138 ?x dblp:author ?y .
139 ?y rdf:type foaf:Person .
140 ?y foaf:name "arvind" .
141
142 # q19
143 ?x dblp:year "2007" .
144 ?x rdfs:label "software" .
145 ?x rdfs:label "time" .
146 ?x rdf:type dblp:Article .
147 ?x dblp:author ?y .
148 ?y rdf:type foaf:Person .
149 ?y foaf:name "zhu" .
150
```

```
151 # q20
152 ?y rdf:type foaf:Person .
153 ?y foaf:name "zhu" .
154 ?y foaf:name "yuntao" .
155
156 # q21
157 ?x dblp:year "2003" .
158 ?x rdfs:label "data" .
159 ?x rdfs:label "content" .
160 ?x rdf:type dblp:Article_in_Proceedings .
161 ?x dblp:author ?y .
162 ?y rdf:type foaf:Person .
163 ?y foaf:name "nikos" .
164
165 # q22
166 ?x rdfs:label "spatial" .
167 ?x rdf:type dblp:Article_in_Proceedings .
168 ?x dblp:author ?y .
169 ?y rdf:type foaf:Person .
170 ?y foaf:name "jignesh" .
171
172 # q23
173 ?x rdfs:label "algorithms" .
174 ?x rdfs:label "parallel" .
175 ?x rdfs:label "spatial" .
176 ?x rdf:type dblp:Article_in_Proceedings .
177 ?x dblp:author ?y .
178 ?x dc:relation "conf" .
179 ?y rdf:type foaf:Person .
180 ?y foaf:name "patel" .
181
182 # q24
183 ?x rdfs:label "implementation" .
184 ?x rdfs:label "evaluation" .
185 ?x rdf:type dblp:Article_in_Proceedings .
186 ?x dblp:last_modified_date "2006-03-31" .
187 ?x dblp:cites ?c .
188 ?x dblp:author ?y .
189 ?y rdf:type foaf:Person .
190 ?y foaf:name "patel" .
191
192 # q25
193 ?x rdfs:label "optimization" .
194 ?x rdfs:label "query" .
195 ?x rdf:type dblp:Article_in_Proceedings .
196 ?x dblp:author ?y .
197 ?x dblp:year "2003" .
198 ?y rdf:type foaf:Person .
199 ?y foaf:name ?n .
200
201 # q26
202 ?x rdfs:label "xml" .
203 ?x rdfs:label "tool" .
```



```
204 ?x rdf:type dblp:Article_in_Proceedings .
205 ?x dblp:year "2004" .
206 ?x dblp:author ?y .
207 ?y rdf:type foaf:Person .
208 ?y foaf:name "patel" .
209
210 # q27
211 ?x rdf:type dblp:Article_in_Proceedings .
212 ?x rdfs:label "architecture" .
213 ?x rdfs:label "web" .
214 ?x dblp:last_modified_date "2005-09-05" .
215 ?x dblp:author ?y .
216 ?y rdf:type foaf:Person .
217 ?y foaf:name "wu" .
218
219 # q28
220 ?x rdf:type dblp:Article_in_Proceedings .
221 ?x rdfs:label "language" .
222 ?x rdfs:label "software" .
223 ?x rdfs:label "system" .
224 ?x dblp:year "2001" .
225 ?x dblp:author ?y .
226 ?y rdf:type foaf:Person .
227 ?y foaf:name "roland" .
228
229 # q29
230 ?x rdf:type dblp:Article_in_Proceedings .
231 ?x rdfs:label "middleware" .
232 ?x dblp:last_modified_date "2006-01-17" .
233 ?x dblp:author ?y .
234 ?y rdf:type foaf:Person .
235 ?y foaf:name "sihvonon" .
236
237 # q30
238 ?x rdf:type dblp:Article_in_Proceedings .
239 ?x rdfs:label "middleware" .
240 ?x rdfs:label "virtual" .
241 ?x dblp:year "2001" .
242 ?x dblp:author ?y .
243 ?y rdf:type foaf:Person .
244 ?y foaf:name "kwang" .
245
246 # q31
247 ?x rdf:type dblp:Article .
248 ?x rdfs:label "java" .
249 ?x rdfs:label "code" .
250 ?x rdfs:label "program" .
251 ?x dblp:author ?y .
252 ?y rdf:type foaf:Person .
253 ?y foaf:name "roland" .
254
255 # q32
256 ?x rdf:type dblp:Article .
```

```
257 ?x rdfs:label "signal" .
258 ?x rdfs:label "space" .
259 ?x dblp:author ?y .
260 ?y rdf:type foaf:Person .
261 ?y foaf:name "zheng" .
262
263 # q33
264 ?x dblp:author ?y .
265 ?y rdf:type foaf:Person .
266 ?y foaf:name "fagin" .
267 ?y foaf:name "roland" .
268
269 # q34
270 ?x dblp:author ?y .
271 ?y rdf:type foaf:Person .
272 ?y foaf:name "zheng" .
273 ?y foaf:name "qui" .
274
275 # q35
276 ?x rdf:type dblp:Article_in_Proceedings .
277 ?x rdfs:label "processing" .
278 ?x rdfs:label "query" .
279
280 # q36
281 ?x rdf:type dblp:Article_in_Proceedings .
282 ?x rdfs:label "xml" .
283 ?x rdfs:label "processing" .
284
285 # q37
286 ?x rdf:type dblp:Article_in_Proceedings .
287 ?x rdfs:label "biological" .
288 ?x rdfs:label "sequence" .
289 ?x dblp:last_modified_date "2007-08-21" .
290 ?x dblp:author ?y .
291 ?y rdf:type foaf:Person .
292 ?y foaf:name "jignesh" .
293
294 # q38
295 ?x rdf:type dblp:Book .
296 ?x rdfs:label "decision" .
297 ?x rdfs:label "intelligent" .
298 ?x rdfs:label "making" .
299 ?x dc:publisher <http://www.springer.de/> .
300
301 # q39
302 ?x rdf:type dblp:Proceedings .
303 ?x rdfs:label "databases" .
304 ?x rdfs:label "biological" .
305 ?x dc:publisher <http://www.springer.de/> .
306
307 # q40
308 ?x rdf:type dblp:Book .
309 ?x rdfs:label "mining" .
```

```
310 ?x rdfs:label "data" .
311
312 # q41
313 ?x rdf:type dblp:Book .
314 ?x rdfs:label "mining" .
315 ?x rdfs:label "data" .
316 ?x dc:publisher <http://www.springer.de/> .
317 ?x dc:relation "trier.de" .
318 ?x dc:relation "books" .
319
320 # q42
321 ?x rdf:type dblp:Book .
322 ?x rdfs:label "intelligence" .
323 ?x rdfs:label "computational" .
324 ?x dc:publisher <http://www.springer.de/> .
325 ?x dc:relation "trier.de" .
326 ?x dblp:year "2007" .
327
328 # q43
329 ?x rdf:type dblp:Book .
330 ?x rdfs:label "biologically" .
331 ?x rdfs:label "inspired" .
332 ?x rdfs:label "methods" .
333
334 # q44
335 ?x rdf:type dblp:Book .
336 ?x rdfs:label "networks" .
337 ?x rdfs:label "neural" .
338
339 # q45
340 ?x rdf:type dblp:Book .
341 ?x rdfs:label "learning" .
342 ?x rdfs:label "machine" .
343 ?x dc:publisher <http://www.springer.de/> .
344
345 # q46
346 ?x rdf:type dblp:Book .
347 ?x rdfs:label "software" .
348 ?x rdfs:label "system" .
349 ?x dc:publisher <http://www.springer.de/> .
350
351 # q47
352 ?x rdf:type dblp:Book .
353 ?x rdfs:label "architecture" .
354 ?x rdfs:label "computer" .
355
356 # q48
357 ?x rdf:type dblp:Book .
358 ?x rdfs:label "web" .
359 ?x dblp:year "2006" .
360 ?x dc:publisher ?p .
361 ?x dblp:editor ?e .
362 ?e foaf:name "kandel" .
```

```

363 ?e foaf:name "abraham" .
364
365 # q49
366 ?x rdf:type dblp:Book .
367 ?x rdfs:label "theoretical" .
368 ?x rdfs:label "science" .
369 ?x dc:publisher <http://www.elsevier.nl/> .
370
371 # q50
372 ?x rdf:type dblp:Book_Chapter .
373 ?x rdfs:label "search" .
374 ?x rdfs:label "semantic" .
375
376 # q51
377 ?x rdf:type dblp:Article .
378 ?x rdfs:label "search" .
379 ?x rdfs:label "concept" .
380 ?x rdfs:label "based" .
381
382 # q52
383 ?x dblp:journal_name "sigmod" .
384 ?x rdf:type dblp:Article .
385 ?x rdfs:label "model" .
386 ?x rdfs:label "information" .
387
388 # q53
389 ?x dblp:journal_name "sigmod" .
390 ?x rdf:type dblp:Article .
391 ?x rdfs:label "dynamic" .
392 ?x rdfs:label "networks" .
393
394 # q54
395 ?x rdf:type dblp:Article_in_Proceedings .
396 ?x rdfs:label "storage" .
397 ?x rdfs:label "adaptive" .
398 ?x dblp:author ?y .
399 ?x dblp:year "2003" .
400 ?y rdf:type foaf:Person .
401 ?y foaf:name "jignesh" .

```

Listing 28: Queries from IMDB benchmark [42].

```

1
2
3 # @PREFIX imdb:
4 # <http://imdb/predicate/> .
5 # @PREFIX imdb_class:
6 # <http://imdb/class/> .
7 # @PREFIX rdf:
8 # <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
9
10 # q1
11 ?x rdf:type imdb_class:name .

```

```
12 ?x imdb:name "washington" .
13 ?x imdb:name "denzel" .
14
15 # q2
16 ?x rdf:type imdb_class:name .
17 ?x imdb:name "eastwood" .
18 ?x imdb:name "clint" .
19
20 # q3
21 ?x rdf:type imdb_class:name .
22 ?x imdb:name "john" .
23 ?x imdb:name "wayne" .
24
25 # q4
26 ?x rdf:type imdb_class:name .
27 ?x imdb:name "smith" .
28 ?x imdb:name "will" .
29
30 # q5
31 ?x rdf:type imdb_class:name .
32 ?x imdb:name "ford" .
33 ?x imdb:name "harrison" .
34
35 # q6
36 ?x rdf:type imdb_class:name .
37 ?x imdb:name "julia" .
38 ?x imdb:name "roberts" .
39
40 # q7
41 ?x rdf:type imdb_class:name .
42 ?x imdb:name "tom" .
43 ?x imdb:name "hanks" .
44
45 # q8
46 ?x rdf:type imdb_class:name .
47 ?x imdb:name "johnny" .
48 ?x imdb:name "depp" .
49
50 # q9
51 ?x rdf:type imdb_class:name .
52 ?x imdb:name "angelina" .
53 ?x imdb:name "jolie" .
54
55 # q10
56 ?x rdf:type imdb_class:name .
57 ?x imdb:name "freeman" .
58 ?x imdb:name "morgan" .
59
60 # q11
61 ?x rdf:type imdb_class:title .
62 ?x imdb:title "gone" .
63 ?x imdb:title "with" .
64 ?x imdb:title "the" .
```

```
65 ?x imdb:title "wind" .
66
67 # q12
68 ?x rdf:type imdb_class:title .
69 ?x imdb:title "wars" .
70 ?x imdb:title "star" .
71
72 # q13
73 ?x rdf:type imdb_class:title .
74 ?x imdb:title "casablanca" .
75
76 # q14
77 ?x rdf:type imdb_class:title .
78 ?x imdb:title "the" .
79 ?x imdb:title "lord" .
80 ?x imdb:title "rings" .
81
82 # q15
83 ?x rdf:type imdb_class:title .
84 ?x imdb:title "the" .
85 ?x imdb:title "sound" .
86 ?x imdb:title "music" .
87
88 # q16
89 ?x rdf:type imdb_class:title .
90 ?x imdb:title "wizard" .
91 ?x imdb:title "oz" .
92
93 # q17
94 ?x rdf:type imdb_class:title .
95 ?x imdb:title "the" .
96 ?x imdb:title "notebook" .
97
98 # q18
99 ?x rdf:type imdb_class:title .
100 ?x imdb:title "forrest" .
101 ?x imdb:title "gump" .
102
103 # q19
104 ?x rdf:type imdb_class:title .
105 ?x imdb:title "the" .
106 ?x imdb:title "princess" .
107 ?x imdb:title "bride" .
108
109 # q20
110 ?x rdf:type imdb_class:title .
111 ?x imdb:title "the" .
112 ?x imdb:title "godfather" .
113
114 # q21
115 ?x imdb:title ?t .
116 ?x rdf:type imdb_class:title .
117 ?x imdb:cast_info ?z .
```

```
118 ?r rdf:type imdb_class:char_name .
119 ?r imdb:name "finch" .
120 ?r imdb:name "atticus" .
121 ?z rdf:type imdb_class:cast_info .
122 ?z imdb:role ?r .
123
124 # q22
125 ?x imdb:title ?t .
126 ?x rdf:type imdb_class:title .
127 ?x imdb:cast_info ?z .
128 ?z rdf:type imdb_class:cast_info .
129 ?r imdb:name "indiana" .
130 ?r imdb:name "jones" .
131 ?z imdb:role ?r .
132 ?r rdf:type imdb_class:char_name .
133
134 # q23
135 ?x imdb:title ?t .
136 ?x rdf:type imdb_class:title .
137 ?x imdb:cast_info ?z .
138 ?z rdf:type imdb_class:cast_info .
139 ?z imdb:role ?r .
140 ?r rdf:type imdb_class:char_name .
141 ?r imdb:name "james" .
142 ?r imdb:name "bond" .
143
144 # q24
145 ?x imdb:title ?t .
146 ?x rdf:type imdb_class:title .
147 ?x imdb:cast_info ?z .
148 ?z rdf:type imdb_class:cast_info .
149 ?z imdb:role ?r .
150 ?r rdf:type imdb_class:char_name .
151 ?r imdb:name "rick" .
152 ?r imdb:name "blaine" .
153
154 # q25
155 ?x imdb:title ?t .
156 ?x imdb:cast_info ?z .
157 ?z rdf:type imdb_class:cast_info .
158 ?z imdb:role ?r .
159 ?r rdf:type imdb_class:char_name .
160 ?r imdb:name "kaine" .
161 ?r imdb:name "will" .
162
163 # q26
164 ?x imdb:title ?t .
165 ?x rdf:type imdb_class:title .
166 ?x imdb:cast_info ?z .
167 ?z rdf:type imdb_class:cast_info .
168 ?z imdb:role ?r .
169 ?r rdf:type imdb_class:char_name .
170 ?r imdb:name "dr." .
```

```
171 ?r imdb:name "hannibal" .
172 ?r imdb:name "lecter" .
173
174 # q27
175 ?x imdb:title ?t .
176 ?x rdf:type imdb_class:title .
177 ?x imdb:cast_info ?z .
178 ?z rdf:type imdb_class:cast_info .
179 ?z imdb:role ?r .
180 ?r rdf:type imdb_class:char_name .
181 ?r imdb:name "norman" .
182 ?r imdb:name "bates" .
183
184 # q28
185 ?x imdb:title ?t .
186 ?x rdf:type imdb_class:title .
187 ?x imdb:cast_info ?z .
188 ?z rdf:type imdb_class:cast_info .
189 ?z imdb:role ?r .
190 ?r rdf:type imdb_class:char_name .
191 ?r imdb:name "darth" .
192 ?r imdb:name "vader" .
193
194 # q29
195 ?x imdb:title ?t .
196 ?x rdf:type imdb_class:title .
197 ?x imdb:cast_info ?z .
198 ?z rdf:type imdb_class:cast_info .
199 ?z imdb:role ?r .
200 ?r rdf:type imdb_class:char_name .
201 ?r imdb:name "the " .
202 ?r imdb:name "wicked" .
203 ?r imdb:name "witch" .
204 ?r imdb:name "the" .
205 ?r imdb:name "west" .
206
207 # q30
208 ?x imdb:title ?t .
209 ?x rdf:type imdb_class:title .
210 ?x imdb:cast_info ?z .
211 ?z rdf:type imdb_class:cast_info .
212 ?z imdb:role ?r .
213 ?r rdf:type imdb_class:char_name .
214 ?r imdb:name "nurse" .
215 ?r imdb:name "ratched" .
216
217 # q31
218 ?x imdb:title ?t .
219 ?x rdf:type imdb_class:title .
220 ?x imdb:movie_info ?i .
221 ?i rdf:type imdb_class:movie_info .
222 ?i imdb:info "frankly" .
223 ?i imdb:info "dear" .
```



```
224 ?i imdb:info "don't" .
225 ?i imdb:info "give" .
226 ?i imdb:info "damn" .
227
228 # q32
229 ?x imdb:title ?t .
230 ?x rdf:type imdb_class:title .
231 ?x imdb:movie_info ?i .
232 ?i rdf:type imdb_class:movie_info .
233 ?i imdb:info "going" .
234 ?i imdb:info "make" .
235 ?i imdb:info "offer" .
236 ?i imdb:info "can't" .
237 ?i imdb:info "refuse" .
238
239 # q33
240 ?x imdb:title ?t .
241 ?x rdf:type imdb_class:title .
242 ?x imdb:movie_info ?i .
243 ?i rdf:type imdb_class:movie_info .
244 ?i imdb:info "understand" .
245 ?i imdb:info "class" .
246 ?i imdb:info "contender" .
247 ?i imdb:info "coulda" .
248 ?i imdb:info "somebody" .
249 ?i imdb:info "instead" .
250 ?i imdb:info "bum" .
251
252 # q34
253 ?x imdb:title ?t .
254 ?x rdf:type imdb_class:title .
255 ?x imdb:movie_info ?i .
256 ?i rdf:type imdb_class:movie_info .
257 ?i imdb:info "toto" .
258 ?i imdb:info "feeling" .
259 ?i imdb:info "not" .
260 ?i imdb:info "kansas" .
261 ?i imdb:info "anymore" .
262
263 # q35
264 ?x imdb:title ?t .
265 ?x rdf:type imdb_class:title .
266 ?x imdb:movie_info ?i .
267 ?i rdf:type imdb_class:movie_info .
268 ?i imdb:info "here's" .
269 ?i imdb:info "looking" .
270 ?i imdb:info "kid" .
271
272 # q36
273 ?x rdf:type imdb_class:title .
274 ?c rdf:type imdb_class:cast_info .
275 ?x imdb:cast_info ?c .
276 ?c imdb:role ?r .
```

```

277 ?r rdf:type imdb_class:char_name .
278 ?x imdb:name "skywalker" .
279 ?c imdb:person ?p .
280 ?p rdf:type imdb_class:name .
281 ?p imdb:name "hamill" .
282
283 # q37
284 ?x imdb:year "2004" .
285 ?x rdf:type imdb_class:title .
286 ?x imdb:title ?t .
287 ?x imdb:cast_info ?c .
288 ?c rdf:type imdb_class:cast_info .
289 ?c imdb:person ?p .
290 ?p rdf:type imdb_class:name .
291 ?p imdb:name "hanks" .
292
293 # q38 #
294 ?r imdb:name ?rn .
295 ?r rdf:type imdb_class:char_name .
296 ?x rdf:type imdb_class:title .
297 ?x imdb:title "yours" .
298 ?x imdb:title "mine" .
299 ?x imdb:title "ours" .
300 ?x imdb:cast_info ?c .
301 ?c rdf:type imdb_class:cast_info .
302 ?c imdb:role ?r .
303 ?c imdb:person ?p .
304 ?p rdf:type imdb_class:name .
305 ?p imdb:name "henry" .
306 ?p imdb:name "fonda" .
307
308 # q39
309 ?x rdf:type imdb_class:title .
310 ?x imdb:title "gladiator" .
311 ?x imdb:cast_info ?c .
312 ?c rdf:type imdb_class:cast_info .
313 ?c imdb:role ?r .
314 ?r imdb:name ?rn .
315 ?r rdf:type imdb_class:char_name .
316 ?c imdb:person ?p .
317 ?p rdf:type imdb_class:name .
318 ?p imdb:name "russell" .
319 ?p imdb:name "crowe" .
320
321 # q40
322 ?x rdf:type imdb_class:title .
323 ?x imdb:title "star" .
324 ?x imdb:title "trek" .
325 ?x imdb:cast_info ?c .
326 ?r rdf:type imdb_class:char_name .
327 ?r imdb:name ?rn .
328 ?c rdf:type imdb_class:cast_info .
329 ?c imdb:role ?r .

```

```

330 ?c imdb:person ?p .
331 ?p rdf:type imdb_class:name .
332 ?p imdb:name "spiner" .
333 ?p imdb:name "brent" .
334
335 # q41
336 ?x imdb:year "1951" .
337 ?x imdb:title ?t .
338 ?x rdf:type imdb_class:title .
339 ?x imdb:cast_info ?c .
340 ?c rdf:type imdb_class:cast_info .
341 ?c imdb:person ?p .
342 ?p rdf:type imdb_class:name .
343 ?p imdb:name "audrey" .
344 ?p imdb:name "hepburn" .
345
346 # q42
347 ?p rdf:type imdb_class:name .
348 ?p imdb:name ?n .
349 ?c imdb:person ?p .
350 ?c rdf:type imdb_class:cast_info .
351 ?c imdb:role ?r .
352 ?r rdf:type imdb_class:char_name .
353 ?r imdb:name "jacques" .
354 ?r imdb:name "clouseau" .
355
356 # q43
357 ?p rdf:type imdb_class:name .
358 ?p imdb:name ?n .
359 ?c imdb:person ?p .
360 ?c rdf:type imdb_class:cast_info .
361 ?c imdb:role ?r .
362 ?r rdf:type imdb_class:char_name .
363 ?r imdb:name "jack" .
364 ?r imdb:name "ryan" .
365
366 # q44
367 ?p rdf:type imdb_class:name .
368 ?p imdb:name "stallone" .
369 ?c imdb:person ?p .
370 ?c rdf:type imdb_class:cast_info .
371 ?c imdb:role ?r .
372 ?r rdf:type imdb_class:char_name .
373 ?r imdb:name "rocky" .
374
375 # q45
376 ?p rdf:type imdb_class:name .
377 ?p imdb:name ?n .
378 ?c imdb:person ?p .
379 ?c rdf:type imdb_class:cast_info .
380 ?c imdb:role ?r .
381 ?r rdf:type imdb_class:char_name .
382 ?r imdb:name "terminator" .

```

```
383
384 # omitted q46 to q49
385
386 # q50
387 ?a rdf:type imdb_class:title .
388 ?a imdb:title "lost" .
389 ?a imdb:title "ark" .
390 ?a imdb:cast_info ?ca .
391 ?ca rdf:type imdb_class:cast_info .
392 ?ca imdb:person ?p .
393 ?p rdf:type imdb_class:name .
394 ?p imdb:name ?n .
395 ?ci rdf:type imdb_class:cast_info .
396 ?ci imdb:person ?p .
397 ?i rdf:type imdb_class:title .
398 ?i imdb:cast_info ?ci .
399 ?i imdb:title "indiana" .
400 ?i imdb:title "jones" .
401 ?i imdb:title "last" .
402 ?i imdb:title "crusade" .
```

A.3 EVALUATION QUERIES FOR CHAPTER 5

In this section, we present the query load that was used during our experiments in Section 5.3. Queries for the SP² benchmark are based on [141], while the DBPSB benchmark queries are generated from seed queries in [123]. All queries are given in RDF NTriples [21] notation.

Listing 29: Prefixes used for SP² and DBPSB queries.

```
1 @prefix rdf:
2   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3 @prefix rdfs:
4   <http://www.w3.org/2000/01/rdf-schema#> .
5 @prefix dc:
6   <http://purl.org/dc/elements/1.1/> .
7 @prefix dcterms:
8   <http://purl.org/dc/terms/> .
9 @prefix xs:
10  <http://www.w3.org/2001/XMLSchema#> .
11 @prefix bench:
12  <http://localhost/vocabulary/bench/> .
13 @prefix foaf:
14  <http://xmlns.com/foaf/0.1/> .
15 @prefix swrc:
16  <http://swrc.ontoware.org/ontology#> .
17 @prefix dbpedia:
18  <http://dbpedia.org/ontology/> .
19 @prefix dbpediapro:
20  <http://dbpedia.org/property/> .
21 @prefix dbpediares:
22  <http://dbpedia.org/resource/> .
23 @prefix skos:
24  <http://www.w3.org/2004/02/skos/core> .
25 @prefix yago:
26  <http://dbpedia.org/class/yago/> .
```

Listing 30: Queries for SP² benchmark [141].

```
1 ### 1
2 ?journal dc:title "Journal 1 (1940)"^^xs:string .
3 ?journal dcterms:issued ?yr .
4 ?journal rdf:type bench:Journal .
5
6 ### 2
7 ?inproc dcterms:partOf ?proc .
8 ?inproc bench:booktitle ?booktitle .
9 ?inproc swrc:pages ?page .
10 ?inproc dc:title ?title .
11 ?inproc rdfs:seeAlso ?ee .
12 ?inproc foaf:homepage ?url .
13 ?inproc dcterms:issued ?yr .
14 ?inproc dc:creator ?author .
```

```
15 ?inproc rdf:type bench:Inproceedings .
16
17 ### 3
18 ?article rdf:type bench:Article .
19 ?article swrc:pages ?value .
20
21 ### 4
22 ?article rdf:type bench:Article .
23 ?article swrc:month ?value .
24
25 ### 5
26 ?article rdf:type bench:Article .
27 ?article swrc:isbn ?value .
28
29 ### 6
30 ?article1 dc:creator ?author1 .
31 ?author1 foaf:name ?name1 .
32 ?article1 swrc:journal ?journal .
33 ?article2 swrc:journal ?journal .
34 ?article2 dc:creator ?author2 .
35 ?author2 foaf:name ?name2 .
36 ?article1 rdf:type bench:Article .
37 ?article2 rdf:type bench:Article .
38
39 ### 7
40 ?article dc:creator ?person .
41 ?person foaf:name ?name .
42 ?person2 foaf:name ?name .
43 ?inproc dc:creator ?person2 .
44 ?article rdf:type bench:Article .
45 ?inproc rdf:type bench:Inproceedings .
46
47 ### 8
48 ?document dcterms:issued ?yr .
49 ?document dc:creator ?author .
50 ?author foaf:name ?name .
51 ?document rdf:type ?class .
52 ?class rdfs:subClassOf foaf:Document .
53
54 ### 9
55 ?doc dc:title ?title .
56 ?bag2 ?member2 ?doc .
57 ?doc2 dcterms:references ?bag2 .
58 ?doc rdf:type ?class .
59 ?class rdfs:subClassOf foaf:Document .
60
61 ### 10
62 ?erdoes foaf:name "Paul Erdoes"^^xs:string .
63 ?document dc:creator ?erdoes .
64 ?document dc:creator ?author .
65 ?document2 dc:creator ?author .
66 ?document2 dc:creator ?author2 .
67 ?author2 foaf:name ?name .
```

```

68 ?erdoes rdf:type foaf:Person .
69
70 ### 11
71 ?person rdf:type foaf:Person .
72 ?subject ?predicate ?person .
73
74 ### 12
75 ?article dc:creator ?person1 .
76 ?person1 foaf:name ?name .
77 ?person2 foaf:name ?name .
78 ?inproc dc:creator ?person2 .
79 ?inproc rdf:type bench:Inproceedings .
80 ?article rdf:type bench:Article .
81
82 ### 13
83 ?erdoes foaf:name "Paul Erdoes"^^xs:string .
84 ?document dc:creator ?erdoes .
85 ?document dc:creator ?author .
86 ?document2 dc:creator ?author .
87 ?document2 dc:creator ?author2 .
88 ?author2 foaf:name ?name .
89 ?erdoes rdf:type foaf:Person .

```

Listing 31: Queries for DBPSB benchmark [123].

```

1 ### 1
2 ?var5 rdf:type dbpedia:Person .
3 ?var5 foaf:page ?var8 .
4 ?var5 dbpedia:thumbnail ?var4 .
5 ?var5 rdfs:label "Thaksin Shinawatra"@mn .
6
7 ### 2
8 ?var5 dbpedia:thumbnail ?var4 .
9 ?var5 rdf:type dbpedia:Person .
10 ?var5 rdfs:label
11     "\u0420\u0438\u0448\u0435,
12     \u0428\u0430\u0440\u0438\u0435\u0440\u0435"@ru .
13 ?var5 foaf:page ?var8 .
14
15 ### 3
16 ?var5 dbpedia:thumbnail ?var4 .
17 ?var5 rdf:type dbpedia:Person .
18 ?var5 rdfs:label
19     "Amadeo, quinto
20     Duque de Aosta"@es .
21 ?var5 foaf:page ?var8 .
22
23 ### 4
24 ?var5 dbpedia:thumbnail ?var4 .
25 ?var5 rdf:type dbpedia:Person .
26 ?var5 rdfs:label "Godeberta"@en .
27 ?var5 foaf:page ?var8 .
28

```

```
29 ### 5
30 ?var5 dbpedia:thumbnail ?var4 .
31 ?var5 rdf:type dbpedia:Person .
32 ?var5 rdfs:label "Thaksin Shinawatra"@nl .
33 ?var5 foaf:page ?var8 .
34
35 ### 6
36 ?var5 dbpedia:thumbnail ?var4 .
37 ?var5 rdf:type dbpedia:Person .
38 ?var5 rdfs:label
39         "\u827E\u9A30\u00B7
40         \u4F0A\u683C\u8A00"@zh .
41 ?var5 foaf:page ?var8 .
42
43 ### 7
44 ?var5 dbpedia:thumbnail ?var4 .
45 ?var5 rdf:type dbpedia:Person .
46 ?var5 rdfs:label "Vlad\u00EDmir Karpets"@es .
47 ?var5 foaf:page ?var8 .
48
49 ### 8
50 ?var5 dbpedia:thumbnail ?var4 .
51 ?var5 rdf:type dbpedia:Person .
52 ?var5 rdfs:label "Daniel Pearl"@en .
53 ?var5 foaf:page ?var8 .
54
55 ### 9
56 ?var5 dbpedia:thumbnail ?var4 .
57 ?var5 rdf:type dbpedia:Person .
58 ?var5 rdfs:label
59         "\u30DE\u30A4\u30C9\u30B9\u30B9
60         \u30A4\u30C9\u30BA\u30B9\u30C9\u30B9
61         \u30EC\u30A2\u30F3"@ja .
62 ?var5 foaf:page ?var8 .
63
64 ### 10
65 ?var5 dbpedia:thumbnail ?var4 .
66 ?var5 rdf:type dbpedia:Person .
67 ?var5 rdfs:label "Walter Hodge"@en .
68 ?var5 foaf:page ?var8 .
69
70 ### 11
71 ?var5 dbpedia:thumbnail ?var4 .
72 ?var5 rdf:type dbpedia:Person .
73 ?var5 rdfs:label "Damian Wayne"@en .
74 ?var5 foaf:page ?var8 .
75
76 ### 12
77 ?var5 dbpedia:thumbnail ?var4 .
78 ?var5 rdf:type dbpedia:Person .
79 ?var5 rdfs:label
80         "\u0417\u0430\u043B
81         \u0435\u0432\u0441\u0430\u043A
```



```

82         \u0438\u0439, \u041A\u0430\u0437
83         \u0438\u043C\u0435\u0436"@ru .
84 ?var5 foaf:page ?var8 .
85
86 ### 13
87 ?var5 dbpedia:thumbnail ?var4 .
88 ?var5 rdf:type dbpedia:Person .
89 ?var5 rdfs:label
90         "\u0413
91         \u043B\u0430\u0443\u0440\u0440\u0438,
92         \u041D\u0438\u0430\u043E\u043B
93         \u043E\u0437 \u0417\u0443\u0443\u0440
94         \u0430\u0431\u043E\u0432\u0432\u0438
95         \u0447"@ru .
96 ?var5 foaf:page ?var8 .
97
98 ### 14
99 ?var5dbpedia:thumbnail ?var4 .
100 ?var5 rdf:type dbpedia:Person .
101 ?var5 rdfs:label "Francis Atterbury"@en .
102 ?var5 foaf:page ?var8 .
103
104 ### 15
105 ?var5 dbpedia:thumbnail ?var4 .
106 ?var5 rdf:type dbpedia:Person .
107 ?var5 rdfs:label "Damian Wayne"@es .
108 ?var5 foaf:page ?var8 .
109
110 ### 16
111 ?var4 dbpediastat:birthPlace
112         "Vigny, Val d'Oise"@en .
113 ?var4 dbpedia:birthDate ?var6 .
114 ?var4 foaf:name ?var8 .
115 ?var4 dbpedia:deathDate ?var10 .
116
117 ### 17
118 ?var4 dbpediastat:birthPlace
119         "Salisbury, England"@en .
120 ?var4 dbpedia:birthDate ?var6 .
121 ?var4 foaf:name ?var8 .
122 ?var4 dbpedia:deathDate ?var10 .
123
124 ### 18
125 ?var4 dbpediastat:birthPlace
126         "Bailey in the city of Durham"@en .
127 ?var4 dbpedia:birthDate ?var6 .
128 ?var4 foaf:name ?var8 .
129 ?var4 dbpedia:deathDate ?var10 .
130
131 ### 19
132 ?var4 dbpediastat:birthPlace
133         "Vasilievskaya,
134         Tambov Governorate,"@en .

```

```
135 ?var4 dbpedia:birthDate ?var6 .
136 ?var4 foaf:name ?var8 .
137 ?var4 dbpedia:deathDate ?var10 .
138
139 ### 20
140 ?var4 dbpediainfo:birthPlace
141         dbpedia:Waltham%2C_Massachusetts .
142 ?var4 dbpedia:birthDate ?var6 .
143 ?var4 foaf:name ?var8 .
144 ?var4 dbpedia:deathDate ?var10 .
145
146 ### 21
147 ?var4 dbpediainfo:birthPlace
148         dbpedia:Valencia%2C_Spain .
149 ?var4 dbpedia:birthDate ?var6 .
150 ?var4 foaf:name ?var8 .
151 ?var4 dbpedia:deathDate ?var10 .
152
153 ### 22
154 ?var4 dbpediainfo:birthPlace
155         dbpedia:Halifax%2C_West_Yorkshire .
156 ?var4 dbpedia:birthDate ?var6 .
157 ?var4 foaf:name ?var8 .
158 ?var4 dbpedia:deathDate ?var10 .
159
160 ### 23
161 ?var4 dbpediainfo:birthPlace dbpedia:Sucre .
162 ?var4 dbpedia:birthDate ?var6 .
163 ?var4 foaf:name ?var8 .
164 ?var4 dbpedia:deathDate ?var10 .
165
166 ### 24
167 ?var4 dbpediainfo:birthPlace
168         dbpedia:L%3BAcar .
169 ?var4 dbpedia:birthDate ?var6 .
170 ?var4 foaf:name ?var8 .
171 ?var4 dbpedia:deathDate ?var10 .
172
173 ### 25
174 ?var4 dbpediainfo:birthPlace
175         dbpedia:%C3%89tampes .
176 ?var4 dbpedia:birthDate ?var6 .
177 ?var4 foaf:name ?var8 .
178 ?var4 dbpedia:deathDate ?var10 .
179
180 ### 26
181 ?var4 dbpediainfo:birthPlace
182         dbpedia:Montgomery_County%2C_Maryland .
183 ?var4 dbpedia:birthDate ?var6 .
184 ?var4 foaf:name ?var8 .
185 ?var4 dbpedia:deathDate ?var10 .
186
187 ### 27
```

```
188 ?var4 dbpedia:prop:birthPlace
189         "Berkeley, Gloucestershire"@en .
190 ?var4 dbpedia:birthDate ?var6 .
191 ?var4 foaf:name ?var8 .
192 ?var4 dbpedia:deathDate ?var10 .
193
194 ### 28
195 ?var4 dbpedia:prop:birthPlace
196         dbpedia:res: Papal_States .
197 ?var4 dbpedia:birthDate ?var6 .
198 ?var4 foaf:name ?var8 .
199 ?var4 dbpedia:deathDate ?var10 .
200
201 ### 29
202 ?var4 dbpedia:prop:birthPlace
203         dbpedia:res: City_of_London .
204 ?var4 dbpedia:birthDate ?var6 .
205 ?var4 foaf:name ?var8 .
206 ?var4 dbpedia:deathDate ?var10 .
207
208 ### 30
209 ?var4 dbpedia:prop:birthPlace
210         "Houghton, Norfolk, England"@en .
211 ?var4 dbpedia:birthDate ?var6 .
212 ?var4 foaf:name ?var8 .
213 ?var4 dbpedia:deathDate ?var10 .
214
215 ### 31
216 ?var4 rdfs:label "(372) Palma"@de .
217 ?var3 skos:broader ?var4 .
218 ?var3 rdfs:label ?var6 .
219
220 ### 32
221 ?var4 rdfs:label "(11554) Asios"@de .
222 ?var3 skos:broader ?var4 .
223 ?var3 rdfs:label ?var6 .
224
225 ### 33
226 ?var4 rdfs:label "(3080) Moisseiev"@de .
227 ?var3 skos:broader ?var4 .
228 ?var3 rdfs:label ?var6 .
229
230 ### 34
231 ?var4 rdfs:label "(1273) Helma"@de .
232 ?var3 skos:broader ?var4 .
233 ?var3 rdfs:label ?var6 .
234
235 ### 35
236 ?var4 rdfs:label
237         "(119878) 2002 CY224"@en .
238 ?var3 skos:broader ?var4 .
239 ?var3 rdfs:label ?var6 .
240
```

```
241 ### 36
242 ?var4 rdfs:label
243     "039A\u578B\u6F5C
244     \u6C34\u8266"@ja .
245 ?var3 skos:broader ?var4 .
246 ?var3 rdfs:label ?var6 .
247
248 ### 37
249 ?var4 rdfs:label
250     "(444) \u042D
251     \u0448\u0435\u0440"@ru .
252 ?var3 skos:broader ?var4 .
253 ?var3 rdfs:label ?var6 .
254
255 ### 38
256 ?var4 rdfs:label
257     "(3834) Zappafrank"@es .
258 ?var3 skos:broader ?var4 .
259 ?var3 rdfs:label ?var6 .
260
261 ### 39
262 ?var4 rdfs:label "(2612) Kathryn"@de .
263 ?var3 skos:broader ?var4 .
264 ?var3 rdfs:label ?var6 .
265
266 ### 40
267 ?var4 rdfs:label "(290) Bruna"@de .
268 ?var3 skos:broader ?var4 .
269 ?var3 rdfs:label ?var6 .
270
271 ### 41
272 ?var4 rdfs:label "(438) Zeuxo"@de .
273 ?var3 skos:broader ?var4 .
274 ?var3 rdfs:label ?var6 .
275
276 ### 42
277 ?var4 rdfs:label "!X\u00F3\u00F5"@de .
278 ?var3 skos:broader ?var4 .
279 ?var3 rdfs:label ?var6 .
280
281 ### 43
282 ?var4 rdfs:label "(1083) Salvia"@de .
283 ?var3 skos:broader ?var4 .
284 ?var3 rdfs:label ?var6 .
285
286 ### 44
287 ?var4 rdfs:label
288     "(1296) Andr\u00E9"@de .
289 ?var3 skos:broader ?var4 .
290 ?var3 rdfs:label ?var6 .
291
292 ### 45
293 ?var1 rdf:type yago:ChristianLGBTPeople .
```

```
294 ?var1 foaf:givenName ?var2 .
295
296 ### 46
297 ?var1 rdf:type yago:DefJamRecordingsArtists .
298 ?var1 foaf:givenName ?var2 .
299
300 ### 47
301 ?var1 rdf:type yago:IndianFilmActors .
302 ?var1 foaf:givenName ?var2 .
303
304 ### 48
305 ?var1 rdf:type yago:EnglishKeyboardists .
306 ?var1 foaf:givenName ?var2 .
307
308 ### 49
309 ?var1 rdf:type yago:GuitarPlayers .
310 ?var1 foaf:givenName ?var2 .
311
312 ### 50
313 ?var1 rdf:type yago:FilipinoFemaleModels .
314 ?var1 foaf:givenName ?var2 .
315
316 ### 51
317 ?var1 rdf:type yago:BluesBrothers .
318 ?var1 foaf:givenName ?var2 .
319
320 ### 52
321 ?var1 rdf:type yago:AmericanSongwriters .
322 ?var1 foaf:givenName ?var2 .
323
324 ### 53
325 ?var1 rdf:type yago:FrenchJazzViolinists .
326 ?var1 foaf:givenName ?var2 .
327
328 ### 54
329 ?var1 rdf:type yago:EnglishJazzComposers .
330 ?var1 foaf:givenName ?var2 .
331
332 ### 55
333 ?var1 rdf:type yago:HarveyMuddCollegeAlumni .
334 ?var1 foaf:givenName ?var2 .
335
336 ### 56
337 ?var1 rdf:type yago:Bassist109842629 .
338 ?var1 foaf:givenName ?var2 .
339
340 ### 57
341 ?var1 rdf:type yago:Curate109983572 .
342 ?var1 foaf:givenName ?var2 .
343
344 ### 58
345 ?var1 rdf:type yago:GreekFemaleModels .
346 ?var1 foaf:givenName ?var2 .
```

```
347
348 ### 59
349 ?var1 rdf:type yago:FilipinoReligiousLeaders .
350 ?var1 foaf:givenName ?var2 .
351
352 ### 60
353 ?var4 skos:subject
354     dbpediares:Category:1004_deaths .
355 ?var4 foaf:name ?var6 .
356
357 ### 61
358 ?var4 skos:subject
359     dbpediares:Category:
360     11th_century_in_England .
361 ?var4 foaf:name ?var6 .
362
363 ### 62
364 ?var4 skos:subject
365     dbpediares:Category:1067_deaths .
366 ?var4 foaf:name ?var6 .
367
368 ### 63
369 ?var4 skos:subject
370     dbpediares:Category:1107_births .
371 ?var4 foaf:name ?var6 .
372
373 ### 64
374 ?var4 skos:subject
375     dbpediares:Category:%C5%Aokoda_trams .
376 ?var4 foaf:name ?var6 .
377
378 ### 65
379 ?var4 skos:subject
380     dbpediares:Category:
381     %C3%81guilas_Cibae%C3%B1as_players .
382 ?var4 foaf:name ?var6 .
383
384 ### 66
385 ?var4 skos:subject
386     dbpediares:Category:1255_births .
387 ?var4 foaf:name ?var6 .
388
389 ### 67
390 ?var4 skos:subject
391     dbpediares:Category:os_BC_births .
392 ?var4 foaf:name ?var6 .
393
394 ### 68
395 ?var4 skos:subject
396     dbpediares:Category:
397     1130_disestablishments .
398 ?var4 foaf:name ?var6 .
399
```

```
400 ### 69
401 ?var4 skos:subject
402     dbpediares:Category:
403     .32_S%26W_Long_firearms .
404 ?var4 foaf:name ?var6 .
405
406 ### 70
407 ?var4 skos:subject
408     dbpediares:Category:1009_deaths .
409 ?var4 foaf:name ?var6 .
410
411 ### 71
412 ?var4 skos:subject
413     dbpediares:Category:1144_deaths .
414 ?var4 foaf:name ?var6 .
415
416 ### 72
417 ?var4 skos:subject
418     dbpediares:Category:1239_deaths .
419 ?var4 foaf:name ?var6 .
420
421 ### 73
422 ?var4 skos:subject
423     dbpediares:Category:1070s_deaths .
424 ?var4 foaf:name ?var6 .
425
426 ### 74
427 ?var4 skos:subject
428     dbpediares:Category:1105 .
429 ?var4 foaf:name ?var6 .
430
431 ### 75
432 ?var3 dbpedia:influenced
433     dbpediares:Rant%C3%B3n_Emeterio_Betances .
434 ?var3 foaf:page ?var4 .
435 ?var3 rdfs:label ?var6 .
436
437 ### 76
438 ?var3 dbpedia:influenced dbpediares:Rob_Corddry .
439 ?var3 foaf:page ?var4 .
440 ?var3 rdfs:label ?var6 .
441
442 ### 77
443 ?var3 dbpedia:influenced
444     dbpediares:Parakrama_Niriella .
445 ?var3 foaf:page ?var4 .
446 ?var3 rdfs:label ?var6 .
447
448 ### 78
449 ?var3 dbpedia:influenced
450     dbpediares:Alexander_VI .
451 ?var3 foaf:page ?var4 .
452 ?var3 rdfs:label ?var6 .
```

```
453
454 ### 79
455 ?var3 dbpedia:influenced dbpediares:Iqbal .
456 ?var3 foaf:page ?var4 .
457 ?var3 rdfs:label ?var6 .
458
459 ### 80
460 ?var3 dbpedia:influenced
461         dbpediares:Al-Maqrizi .
462 ?var3 foaf:page ?var4 .
463 ?var3 rdfs:label ?var6 .
464
465 ### 81
466 ?var3 dbpedia:influenced
467         dbpediares:Clarence_Irving_Lewis .
468 ?var3 foaf:page ?var4 .
469 ?var3 rdfs:label ?var6 .
470
471 ### 82
472 ?var3 dbpedia:influenced
473         dbpediares:Ibn_Khaleel .
474 ?var3 foaf:page ?var4 .
475 ?var3 rdfs:label ?var6 .
476
477 ### 83
478 ?var3 dbpedia:influenced
479         dbpediares:David_Friedl%C3%A4nder .
480 ?var3 foaf:page ?var4 .
481 ?var3 rdfs:label ?var6 .
482
483 ### 84
484 ?var3 dbpedia:influenced
485         dbpediares:John_Warnock .
486 ?var3 foaf:page ?var4 .
487 ?var3 rdfs:label ?var6 .
488
489 ### 85
490 ?var3 dbpedia:influenced
491         dbpediares:Vladimir_Lenin .
492 ?var3 foaf:page ?var4 .
493 ?var3 rdfs:label ?var6 .
494
495 ### 86
496 ?var3 dbpedia:influenced
497         dbpediares:Niall_McLaren .
498 ?var3 foaf:page ?var4 .
499 ?var3 rdfs:label ?var6 .
500
501 ### 87
502 ?var3 dbpedia:influenced
503         dbpediares:David_J._Farber .
504 ?var3 foaf:page ?var4 .
505 ?var3 rdfs:label ?var6 .
```



```
506 ### 88
507 ?var3 dbpedia:influenced
508     dbpediares:Fran_Lebowitz .
509 ?var3 foaf:page ?var4 .
510 ?var3 rdfs:label ?var6 .
511
512 ### 89
513 ?var3 dbpedia:influenced
514     dbpediares:Kathleen_Raine .
515 ?var3 foaf:page ?var4 .
516 ?var3 rdfs:label ?var6 .
517
518 ### 90
519 ?varo rdfs:label "The Subtle Knife"@en .
520 ?varo rdf:type ?var1 .
521
522 ### 91
523 ?varo rdfs:label "Patrioter"@sv .
524 ?varo rdf:type ?var1 .
525
526 ### 92
527 ?varo rdfs:label "Scar Tissue (libro)"@es .
528 ?varo rdf:type ?var1 .
529
530 ### 93
531 ?varo rdfs:label
532     "Jason Bournes ultimatum"@nn .
533 ?varo rdf:type ?var1 .
534
535 ### 94
536 ?varo rdfs:label "Jane Eyre"@fr .
537 ?varo rdf:type ?var1 .
538
539 ### 95
540 ?varo rdfs:label
541     "Gone with the Wind (livro)"@pt .
542 ?varo rdf:type ?var1 .
543
544 ### 96
545 ?varo rdfs:label "Alkumets\u00E4"@fi .
546 ?varo rdf:type ?var1 .
547
548 ### 97
549 ?varo rdfs:label
550     "Flight from the Dark"@en .
551 ?varo rdf:type ?var1 .
552
553 ### 98
554 ?varo rdfs:label "Mongol Empire"@en .
555 ?varo rdf:type ?var1 .
556
557 ### 99
558
```

```
559 ?varo rdfs:label "Kultahattu"@fi .
560 ?varo rdf:type ?var1 .
561
562 ### 100
563 ?varo rdfs:label
564     "Aseiden k\u00E4ytt\u00F6"@fi .
565 ?varo rdf:type ?var1 .
566
567 ### 101
568 ?varo rdfs:label "Marrow (novel)"@en .
569 ?varo rdf:type ?var1 .
570
571 ### 102
572 ?varo rdfs:label "The Acid House"@en .
573 ?varo rdf:type ?var1 .
574
575 ### 103
576 ?varo rdfs:label "\u6B63\u4E49\u8BBA"@zh .
577 ?varo rdf:type ?var1 .
578
579 ### 104
580 ?varo rdfs:label "Dawn of the Dragons"@en .
581 ?varo rdf:type ?var1 .
582
583 ### 105
584 ?var2 rdf:type dbpedia:Person .
585 ?var2 rdfs:label
586     "Ab\u016B l-Hasan Ban\u012Bsadr"@de .
587 ?var2 foaf:page ?var4 .
588
589 ### 106
590 ?var2 rdf:type dbpedia:Person .
591 ?var2 rdfs:label
592     "Abdul Rahman of Negeri Sembilan"@en .
593 ?var2 foaf:page ?var4 .
594
595 ### 107
596 ?var2 rdf:type dbpedia:Person .
597 ?var2 rdfs:label "A.W. Farwick"@en .
598 ?var2 foaf:page ?var4 .
599
600 ### 108
601 ?var2 rdf:type dbpedia:Person .
602 ?var2 rdfs:label "Abdullah G\u00FCl"@sv .
603 ?var2 foaf:page ?var4 .
604
605 ### 109
606 ?var2 rdf:type dbpedia:Person .
607 ?var2 rdfs:label "Aaron Pe\u00F1a"@en .
608 ?var2 foaf:page ?var4 .
609
610 ### 110
611 ?var2 rdf:type dbpedia:Person .
```

```
612 ?var2 rdfs:label "Abby Lockhart"@fr .
613 ?var2 foaf:page ?var4 .
614
615 ### 111
616 ?var2 rdf:type dbpedia:Person .
617 ?var2 rdfs:label "Abd al-Latif"@en .
618 ?var2 foaf:page ?var4 .
619
620 ### 112
621 ?var2 rdf:type dbpedia:Person .
622 ?var2 rdfs:label "Abdel Halim Khaddam"@fr .
623 ?var2 foaf:page ?var4 .
624
625 ### 113
626 ?var2 rdf:type dbpedia:Person .
627 ?var2 rdfs:label "Abdur Rahman Khan"@fr .
628 ?var2 foaf:page ?var4 .
629
630 ### 114
631 ?var2 rdf:type dbpedia:Person .
632 ?var2 rdfs:label
633     "A\u0142\u0142a Kudriawcewa"@pl .
634 ?var2 foaf:page ?var4 .
635
636 ### 115
637 ?var2 rdf:type dbpedia:Person .
638 ?var2 rdfs:label "Aaron Raper"@en .
639 ?var2 foaf:page ?var4 .
640
641 ### 116
642 ?var2 rdf:type dbpedia:Person .
643 ?var2 rdfs:label "A.L. Williams"@en .
644 ?var2 foaf:page ?var4 .
645
646 ### 117
647 ?var2 rdf:type dbpedia:Person .
648 ?var2 rdfs:label "Abdul Kadir Khan"@sv .
649 ?var2 foaf:page ?var4 .
650
651 ### 118
652 ?var2 rdf:type dbpedia:Person .
653 ?var2 rdfs:label "A. J. Pierzynski"@en .
654 ?var2 foaf:page ?var4 .
655
656 ### 119
657 ?var2 rdf:type dbpedia:Person .
658 ?var2 rdfs:label "Abdullah Ahmad Badawi"@pl .
659 ?var2 foaf:page ?var4 .
660
661 ### 120
662 ?var2 rdf:type dbpedia:Person .
663 ?var2 rdfs:label
664     "Abdelbaset Ali Mohmed Al Megrahi"@en .
```

