Strategies for Managing Linked Enterprise Data

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

von
Mikhail Galkin
aus
Uljanowsk, Russland

Bonn, 30.08.2018

1. Gutachter:	Prof. Dr. Sören Auer
2. Gutachter:	Dr. Dmitry Mouromtsev
Tag der Promotion:	10.12.2018
Erscheinungsjahr:	2019

Abstract

Data, information and knowledge become key assets of our 21st century economy. As a result, data and knowledge management become key tasks with regard to sustainable development and business success. Often, knowledge is not explicitly represented residing in the minds of people or scattered among a variety of data sources. Knowledge is inherently associated with semantics that conveys its meaning to a human or machine agent. The Linked Data concept facilitates the semantic integration of heterogeneous data sources. However, we still lack an effective knowledge integration strategy applicable to enterprise scenarios, which balances between large amounts of data stored in legacy information systems and data lakes as well as tailored domain specific ontologies that formally describe real-world concepts.

In this thesis we investigate strategies for managing linked enterprise data analyzing how actionable knowledge can be derived from enterprise data leveraging knowledge graphs. Actionable knowledge provides valuable insights, supports decision makers with clear interpretable arguments, and keeps its inference processes explainable. The benefits of employing actionable knowledge and its coherent management strategy span from a holistic semantic representation layer of enterprise data, i.e., representing numerous data sources as one, consistent, and integrated knowledge source, to unified interaction mechanisms with other systems that are able to effectively and efficiently leverage such an actionable knowledge. Several challenges have to be addressed on different conceptual levels pursuing this goal, i.e., means for representing knowledge, semantic data integration of raw data sources and subsequent knowledge extraction, communication interfaces, and implementation.

In order to tackle those challenges we present the concept of Enterprise Knowledge Graphs (EKGs), describe their characteristics and advantages compared to existing approaches. We study each challenge with regard to using EKGs and demonstrate their efficiency. In particular, EKGs are able to reduce the semantic data integration effort when processing large-scale heterogeneous datasets. Then, having built a consistent logical integration layer with heterogeneity behind the scenes, EKGs unify query processing and enable effective communication interfaces for other enterprise systems. The achieved results allow us to conclude that strategies for managing linked enterprise data based on EKGs exhibit reasonable performance, comply with enterprise requirements, and ensure integrated data and knowledge management throughout its life cycle.

Автореферат

Актуальность. Объем используемых компаниями данных и формализованных знаний, значительно возрос в последние годы, сделав их ключевыми компонентами экономики двадцать первого века. Таким образом, управление данными и знаниями является ключевой задачей устойчивого развития и делового успеха. Как правило, знания не представлены явным и структурированным образом в информационных системах, а воплощены в форме опыта сотрудников или распределены среди множества источников данных. Смысл знаний может быть представлен в виде формальной семантики, которую необходимо передавать интеллектуальным агентам для понимания знаний. Концепция Связанных данных (Linked Data) способствует развитию семантической интеграции гетерогенных источников данных за счет определения уникальных идентификаторов абстрактных и реальных сущностей, а также ссылок между такими идентификаторами, которые в совокупности составляют граф понятий и связей между ними. Однако, существующие механизмы интеграции данных и знаний не приспособлены к корпоративным сценариям использования, где большие объемы данных информационных систем могут содержаться вместе со специализированными онтологиями предметных областей, которые формально описывают сущности реального мира. Таким образом, актуальной проблемой является организация управления разнородными корпоративными данными, которая позволит эффективно извлекать и применять содержащиеся в данных знания для поддержки принятия решений.

В данной работе исследуются стратегии управления связанными корпоративными данными, которые позволяют выделять полезные и действенные в практическом плане знания из корпоративных данных с помощью графов знаний. Под действенными знаниями понимаются знания, которые поддерживают процесс принятия решений четкими и интерпретируемыми аргументами, подходящими для логического вывода. Преимущества внедрения и использования действенных знаний и соответствующих стратегий управления ими охватывают как целостное семантическое представление корпоративных данных в логически единый источник, так и механизмы взаимодействия с другими информационными системами, способными эффективно использовать действенные знания.

Целью работы является исследование применимости семантических технологий для создания стратегии управления связанными корпоративными данными, использующей действенные знания из больших объемов гетерогенных корпоративных данных. Для достижения поставленной цели были сформулированы следующие задачи:

1. средства представления знаний;
2. способы семантической интеграции гетерогенных источников данных с последующим выделением знаний их них;
3. способы эффективной обработки запросов к таким знаниям, так как осуществление и исполнение запросов является одним из основным механизмов взаимодействия информационных систем.

Для решения поставленных задач в данной работе формулируется концепция Корпоративных Графов Знаний (КГЗ), анализируются характеристики и преимущества такого подхода к организации знаний в сравнении с существующими подходами. Так, КГЗ способны упростить процесс семантической интеграции крупномасштабных гетерогенных наборов и источников данных. В работе предложены способы построения логического интеграционного слоя на основе крупномасштабных гетерогенных наборов. КГЗ позволяют унифицировать процесс обработки и исполнения запросов, обеспечивая эффективные механизмы для коммуникации с корпоративными информационными системами.

Новые научные результаты:

1. Концепция Корпоративных Графов Знаний, служащий связующим элементом между гетерогенными корпоративными данными и действенными корпоративными знаниями.

2. Методы и способы организации корпоративных знаний согласно семантической графовой модели Resource Description Framework (RDF), логической интеграции данных и выделения знаний с помощью свойства полуструктурированости модели RDF.

3. Универсальный подход к обработке запросов к полученному графу с помощью усовершенствованных и оптимизированных алгоритмов планирования и исполнения запросов на языке SPARQL.

Достигнутые результаты позволяют сделать вывод, что стратегии управления связанными корпоративными данными на основе КГЗ обладают высоким потенциалом и демонстрируют высокую производительность, соблюдение корпоративных сценариев использования и соответствующих требований, а также всестороннюю поддержку жизненного цикла данных и знаний.

Acknowledgements

Throughout the exciting Ph.D. journey I met a number of people who inspired and supported me all those years. I would like to thank Prof. Dr. Sören Auer for giving me a chance to pursue the degree in Germany at the Enterprise Information Systems department at the University of Bonn. He also motivated me for applying for a DAAD scholarship which we successfully received. His supervision and advice fostered the whole research. Furthermore, I would like to thank Prof. Dr. Maria-Esther Vidal for her invaluable help, inspiration to do academic research, fruitful scientific discussions, many papers written together, and keeping me up last years of the Ph.D. term! Moreover, I would like to thank my advisors Dr. Simon Scerri and Dr. Fabrizio Orlandi for their patience when introducing me to everyday scientific routines and teaching, and for general advice how things work here in academia. Prof. Dr. Jens Lehmann, Dr. Christoph Lange, Dr. Ioanna Lytra helped a lot addressing details with DAAD and double-degree papers so that I would like to thank them for carrying this tedious work and their endless patience. Special thanks goes to Diego "El Romántico" Collarana, Irlán "El Chamaco" Grangel-González, Vitalis Wiens, Niklas Petersen, Najmeh Mousavi, Camilo Morales, and Matthew English for their academic support this thesis could not be completed without.

I cannot fully express my sincere gratitude to Dr. Dmitry Mouromtsev from ITMO University he truly deserves as a person who introduced me to Semantic Web, Artificial Intelligence, scientific mindset and science in general during undergraduate years, supported my first attempts of scientific writing, made it possible to participate in the double-degree Ph.D. program, and kindly participated in solving various issues appeared at the ITMO side.

Last, but not least, I would like to thank a good old friend of mine Vitaly Kandyba whose cheerfulness and passion motivated me to achieve better and better results and who never let me down all those years, my family and friends for their unconditional support, members of Fraunhofer Campus Classic Rock band for sharing wisdoms of life, and Elena Filbert for being such an awesome person.

Contents

CHAPTER 1

Introduction

1.1 Motivation

In our increasingly digitized world data and formally represented knowledge have become key assets. Nowadays, a vast variety of activities involve data generation, interchange, storage, and application. Data is of the utmost priority for numerous companies which business processes could not be performed without various forms of data processing. Enterprise data, therefore, is a source of high revenues on one hand, and reason of injurious losses, on the other hand. The scale enterprise data is operated at implies a significant potential to be fulfilled and impact of such data on the decision making process, e.g., from manufacturing and supply chains to precision medicine and drug discovery. Truly, enterprise data is capable of tackling massive and long-term challenges. Enterprise data management becomes, in turn, a key component to get an advantage over market competitors. The more knowledge one can extract from the data and subsequently employ, the better market strategy can be derived. However, the majority of currently available enterprise data management approaches are not able to leverage knowledge encoded in the data.

The attributes of enterprise data, such as heterogeneity, diversity, measurability, quality, security, characterize as well the World Wide Web (Web) that emerged into a "Web of Data". As enterprise data is held privately and not available for direct research and experiments, we project its features on Web data and study it applying the findings in the enterprise domain. In contrast to the first versions of the Web, i.e., "Web of Documents" that contained at most HyperText Markup Language (HTML) documents and links between them, the Web of Data goes deeper and includes abstract concepts, facts, and real-world entities in a giant graph-like structure. The Semantic Web, coined by Tim Berners-Lee [1], now a standard of the World Wide Web Consortium (W3C)[1], aims at enriching the Web of Data with semantics understandable by machines. W3C proposed a set of principles known as Linked Data in order to facilitate the dissemination of the Semantic Web. Linked Data promotes semantic technologies, i.e., Resource Description Framework (RDF) as a *lingua franca* for knowledge representation, SPARQL as a query language for RDF, and Web Ontology Language (OWL) as a logical formalism to encode ontologies. Ontologies are used to create a basic, logical, machine-readable description of concepts and entities in a chosen domain of discourse. With the above-mentioned features and descriptions, we consider semantic datasets available on the Web as a decent estimate of enterprise data perplexities.

Semantic technologies are envisioned for extracting knowledge from the raw data sources and forming semantic networks, and sharing knowledge via communication interfaces. Knowledge graph (KG) is a novel knowledge organization paradigm that caters for all stages of knowledge management, i.e., from

[1] https://www.w3.org/standards/semanticweb/

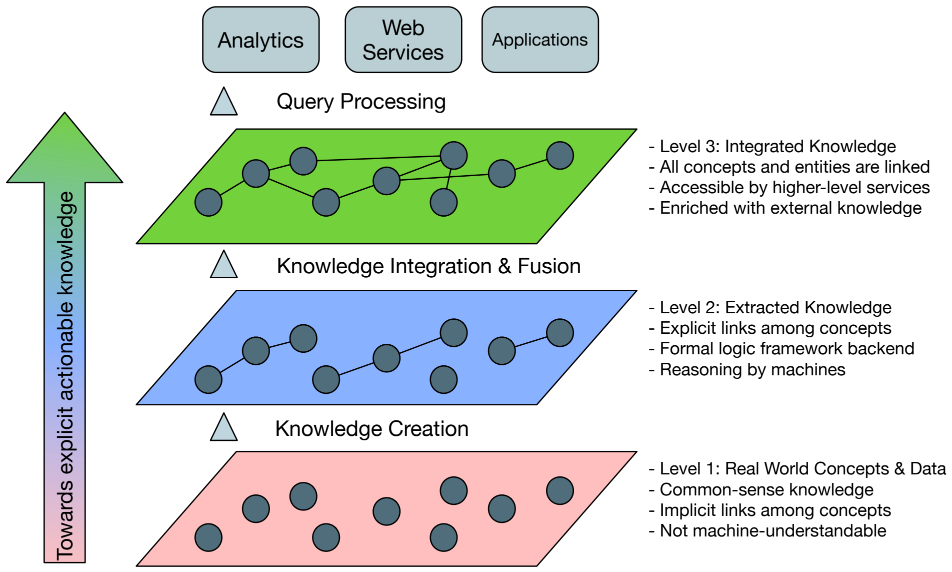

Analytics
Web Services
Applications
Query Processing
- Level 3: Integrated Knowledge
- All concepts and entities are linked
- Accessible by higher-level services
- Enriched with external knowledge
Knowledge Integration & Fusion
- Level 2: Extracted Knowledge
- Explicit links among concepts
- Formal logic framework backend
- Reasoning by machines
Knowledge Creation
- Level 1: Real World Concepts & Data
- Common-sense knowledge
- Implicit links among concepts
- Not machine-understandable
Towards explicit actionable knowledge

Figure 1.1: **Stages towards integrating data into knowledge graphs.** Level 1 consists of real world concepts and common-sense knowledge presented mostly in unstructured formats. The task of knowledge extraction and creation leads to Level 2. Extracted knowledge resides within Level 2 and shares a formal representation technique but is still not interlinked and integrated. The knowledge integration and fusion task supports Level 3 with the integrated knowledge which is then leveraged by external services.

raw data processing to leveraging knowledge in certain applications. At a larger scale Big Data generates Big Knowledge. Big Knowledge retains characteristics of Big Data, e.g., volume, velocity, variety, and adds semantic integration of extracted knowledge as an intermediate layer between large-scale heterogeneous data and sophisticated applications that require deep analysis of inherent knowledge.

Being applied to enterprise data, semantics and Linked Data establish a new concept of Linked Enterprise Data (LED) that aims at extracting, organizing, and processing viable knowledge out of the input data. Therefore, LED is capable of providing an added value to the enterprise data as an asset.

This thesis studies strategies for managing Linked Enterprise Data by involving knowledge graphs thus introducing Enterprise Knowledge Graphs as a new framework for making knowledge explicit.

1.2 Problem Statement and Challenges

The amount of data either created manually or generated automatically grows every year. However, there still exists a noticeable gap between collected data and its application opportunities. Just the fact one has more data does not imply more revenue or better service quality. More data encodes more inherent facts and insights that have to be made explicit (for both humans and machines) in order to properly understand and analyze them. Therefore, this thesis contributes to the industrial and technological challenge of *creating actionable knowledge from Big Enterprise Data*.

The approach that aims to address the problem has to span multiple stages of data processing as shown in Figure 1.1. The first level comprises real-world concepts and data itself with implicit facts. The coming data in original formats is often not machine-understandable. Hence the first stage (and *Challenge 1*) consists in extracting, modeling and representing knowledge encoded in the source data. The second level incorporates the extracted knowledge in a machine-readable format, e.g., RDF. Links among concepts and entities within one dataset become externalized. Nevertheless, the level still exhibits low connectivity among those datasets, i.e., original datasets have to be linked between each other. The task of knowledge integration and fusion constitutes the second stage (and *Challenge 2*) of the envisioned approach. The third level presents a unified view on the knowledge. All internal datasets are interlinked to an extent that allows higher-level services to query an integrated level as a black box, i.e., logically one source of all available knowledge. In turn, the third stage (*Challenge 3*) implies effective and efficient query processing. Once the querying mechanisms are settled, knowledge analysis and utilization by other applications can be performed. In other words, knowledge is actionable when certain meaningful steps can be taken based on newly obtained insights. Eventually, having passed all the described stages knowledge that was extracted from heterogeneous, large-scale data becomes truly explicit and actionable for both machines and humans.

As the main problem is much larger than it can be seen from the above descriptions and can not be easily solved with one thesis, we leave out of the thesis scope numerous tasks and challenges, e.g., physical data storage infrastructures and network architectures, graph databases, governance and maturity models, description logics for knowledge graph ontologies, programming and user interfaces, and particular applications. We are convinced that new findings presented in this thesis would serve a promising basis for a future work addressing those out of the scope challenges.

1.2.1 Challenges for Knowledge Generation from Big Data

Challenge 1: Representing knowledge obtained from Big Data.

There exist numerous paradigms for organizing relational and non-relational databases, e.g., data warehouse [2], data lake [3], that rely mostly on a database schema. In contrast, there is no standardized and embraced by the community knowledge organization model. Selecting the most relevant knowledge modeling and representation mechanism defines subsequent restrictions on formats, extent of machine-readability and reasoning, as well as possible communication interfaces. The amount of options greatly varies from conceptual-modeling-inspired UML[2] or IDEF[3] family to formal logic-inspired approaches such as rules, frames, and semantic networks.

Challenge 2: Integrating knowledge.

Knowledge integration aims at linking disparate datasets, fusing similar concepts and entities, enriching them with externally available knowledge. In the context of Big Data and Big Knowledge integration takes place both at physical and logical levels, i.e., physical integration implies distribution and partitioning of original knowledge sources, whereas logical integration involves model and schema processing. Entity disambiguation and linking has to be applied in order to determine resources relatedness with their subsequent fusion to get a unified representation of a certain entity without duplicates.

Challenge 3: Effective and efficient query processing.

The width of knowledge application spectrum is largely defined by the querying mechanism, its expressivity and capabilities. Query processing has to adhere to two principles, i.e., efficiency and effectiveness. The principle of efficient query processing consists in minimization of query cost parameters, typically execution time, and compliance with query complexity threshold that can be evaluated by the query engine. The principle of effective query processing stands for completeness and correctness of query

[2]Unified Modeling Language `http://www.omg.org/spec/UML/About-UML/`
[3]Integration DEFinition `http://www.idef.com/`

answers after evaluating by the query engine. Completeness and correctness are especially important metrics when processing large-scale distributed or federated environments.

1.3 Research Questions

Based on the challenges, we derive the following research questions to be addressed in the thesis:

Research Question 1 (RQ1)

How can we apply the concept of knowledge graphs for representing enterprise information to become actionable knowledge?

Graph-based models are inherently semi-structured [4]. To answer this question we analyze why semantic technologies can serve as a comprehensive basis for building a knowledge model for linked enterprise data. Particularly, we adopt a concept of knowledge graphs and demonstrate its benefits compared to traditional knowledge management approaches. Furthermore, we investigate data semantification pipelines, i.e., ways of transforming source data to semantic formats, e.g., RDF.

Research Question 2 (RQ2)

How can existing non-semantic data sources be lifted and semantically integrated leveraging enterprise knowledge graphs?

In this question, we study semantic data integration and fusion techniques for knowledge graphs. Knowledge graphs extracted from raw data often exhibit a high extent of heterogeneity, especially at the schema level. We revise semantic similarity mechanisms to address heterogeneous integration and find resources that are logically equivalent. Additionally, we define techniques and policies for fusing, i.e., merging, equivalent entities to create a unified knowledge representation without duplicates.

Research Question 3 (RQ3)

How can we perform semantic query processing of federated Enterprise Data efficiently?

Querying of knowledge graphs is investigated in the third research question. Due to increasing volumes of data and knowledge centralized approaches become less applicable than distributed and federated. We thus overview existing pitfalls in semantic query processing of large-scale federated infrastructures, i.e., query decomposition, source selection, query execution, join operators.

1.4 Thesis Overview

In order to present a high-level but descriptive overview of the achieved results during the conducted research, we emphasize the main contributions of the thesis and provide references to scientific articles covering these contributions published throughout the whole term.

1.4.1 Contributions

Contributions for RQ1

Enterprise Knowledge Graph as a formal framework for organizing linked enterprise data.

We demonstrate why knowledge graphs are more effective for knowledge management. From a family of knowledge graphs for various domains we then devise a formal framework, i.e., *Enterprise Knowledge Graph (EKG)*, for modeling and representing knowledge extracted from enterprise data. EKG lays solid foundations for the following knowledge integration process on the one hand while considering a set of enterprise-related requirements on the other hand. We perform a comprehensive evaluation and comparison of EKGs with existing knowledge management frameworks.

Contributions for RQ2

A computational framework for semantically integrating heterogeneous knowledge graphs.

A typical graph integration framework pipeline includes knowledge extraction, actual graph integration and interlinking steps. As a part of the framework we employ a notion of subgraphs that comprise all available facts about a resource in the knowledge graph. We describe an approach for extracting knowledge from web tables, one of the most popular means for representing enterprise data, thus reminiscing a connection between web and enterprise data. We adopt a novel RDF graph integration technique that tackles semantics encoded in RDF graphs and incorporates data semantification pipelines, semantic similarity functions, and fusion policies. Then, for interlinking we study how this technique accommodates a newly defined semantic join operator which applies fusion policies for merging identified similar subgraphs. The physical semantic join operator able to identify similar entities within heterogeneous sources while processing a query constitutes the main contribution for this research question. More high-level integration frameworks are then able to employ the operator results for knowledge deduplication and fusion.

Contributions for RQ3

A set of algorithms for performing and optimizing federated query processing over knowledge graphs.

Due to increasing data volumes knowledge graphs tend to be deployed in distributed or federated environments. In contrast to centralized database engines, querying a federated environment poses additional challenges for query decomposition, planning, and execution. We again leverage the notion of RDF subgraphs in order to present a unified logical view on the federated knowledge graph. Given that, we adopt novel algorithms for query decomposition and planning. We perform an extensive evaluation of the algorithms against state-of-the-art federated query engines and prove that the algorithms perform faster while keeping full completeness of answers. As the main contribution, we come up with the new algorithm for a multi-way join operator for querying RDF knowledge graphs using SPARQL query language. We demonstrate advantages of multi-way join operators at query planning and execution stages compared to widely used binary join operators. Finally, we tackle the problem of query optimization by introducing algorithms for query planning that are able to combine both binary and multi-way operators in one query plan thus accelerating performance of a subset of complex queries.

1.4.2 Publications

The following publications constitute a scientific basis of this thesis and serve as a reference point for numerous figures, tables and ideas presented in the later chapters:

1. **Mikhail Galkin**, Dmitry Mouromtsev, Sören Auer. *Identifying Web Tables: Supporting a Neglected Type of Content on the Web.* In Proceedings of the 6th International Conference on Knowledge Engineering and Semantic Web (KESW) 2015, 48–62, Springer;

2. **Mikhail Galkin**, Sören Auer, Simon Scerri. *Enterprise Knowledge Graphs: A Backbone of Linked Enterprise Data.* In Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence (WI) 2016, 497–502, ACM;

3. **Mikhail Galkin**, Sören Auer, Maria-Esther Vidal, Simon Scerri. *Enterprise Knowledge Graphs: A Semantic Approach for Knowledge Management in the Next Generation of Enterprise Information Systems.* In Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS) 2017, Volume 2, 88–98, Scitepress;

4. Diego Collarana, **Mikhail Galkin**, Ignacio Traverso-Ribón, Christoph Lange, Maria-Esther Vidal, Sören Auer. *MINTE: semantically integrating RDF graphs.* In Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics (WIMS) 2017, 22:1–22:11, ACM. This is a joint work with Diego Collarana, another PhD student at the University of Bonn. In this paper, my contributions include preparing a motivating example, contributions to a overall problem definition, formalization of fusion policies, complexity analysis, preparation of datasets for experiments, and reviewing related work.

5. **Mikhail Galkin**, Diego Collarana, Ignacio Traverso-Ribón, Christoph Lange, Maria-Esther Vidal, Sören Auer. *SJoin: A Semantic Join Operator to Integrate Heterogeneous RDF Graphs.* In Proceedings of the 28th International Conference of Database and Expert Systems Applications (DEXA) 2017, 206–221, Springer. This is a joint work with Diego Collarana, another PhD student at the University of Bonn. In this paper, I contributed to the definition and implementation of the semantic join operator, problem formalization and complexity analysis, preparation of datasets for experiments, evaluation and analysis of obtained results, and reviewing related work.

6. **Mikhail Galkin**, Kemele M. Endris, Maribel Acosta, Diego Collarana, Maria-Esther Vidal, Sören Auer. *SMJoin: A Multi-way Join Operator for SPARQL Queries.* In Proceedings of the 13th International Conference on Semantic Systems (SEMANTiCS) 2017, 104–111, Springer. This is a joint work with Kemele M. Endris, a PhD student from Leibniz University in Hannover, and Diego Collarana, a PhD student at the University of Bonn. In this paper, I contributed to the definition, description, and implementation of the multi-way join operator, problem formalization, preparation of datasets for experiments, analysis of obtained results, and reviewing related work.

7. **Mikhail Galkin**, Maria-Esther Vidal, Sören Auer. *Towards a Multi-way Similarity Join Operator*. In Proceedings of the 21st European Conference on Advances in Databases and Information Systems (ADBIS) 2017 Short Papers and Workshops, Springer;

8. Kemele M. Endris, **Mikhail Galkin**, Ioanna Lytra, Mohamed Nadjib Mami, Maria-Esther Vidal, Sören Auer. *MULDER: Querying the Linked Data Web by Bridging RDF Molecule Templates*. Accepted for publication in the LNCS Transactions on Large-Scale Data- and Knowledge-Centered Systems Journal (Transactions LDKS), 2018. This is a joint work with Kemele M. Endris, a PhD student from Leibniz University in Hannover, and Mohamed Nadjib Mami, a PhD student at the University of Bonn. In this paper, I contributed to the motivating example, definition of algorithms, visual description of the proposed approach, preparation of infrastructure and datasets for experiments, and reviewing related work.

The whole list of publications completed during the PhD term is available in Appendix A.

1.5 Thesis Structure

The thesis is structured into eight chapters. Chapter 1 introduces the thesis covering the main research problem, motivation for the conducted study, research questions, scientific contributions that address research questions, and a list of published scientific papers that formally describe those contributions.

Chapter 2 presents fundamental concepts and preliminaries in the fields of Semantic Web, Linked Data, Data Integration, Knowledge Graphs necessary for understanding the research problem and chosen approaches to tackle this problem.

Chapter 3 discusses state-of-the-art community efforts in various domains, e.g., knowledge graphs, semantic data integration, knowledge processing, aimed at making explicit linked knowledge out of the heterogeneous real-world data.

In Chapter 4, we define a key concept for the thesis, i.e., Enterprise Knowledge Graphs, and describe a comprehensive framework for creating such graphs. That is, starting with the basic definitions, we derive various subclasses of knowledge graphs, their functional and non-functional requirements, and present a unified knowledge graph assessment framework.

Chapter 5 reports the efforts aimed at physical and logical knowledge integration and fusion. We show how ontology-based data integration is able to enrich the quality of a knowledge graph, what are the advantages of semantic similarity functions and benefits of various knowledge fusion strategies. In this chapter, we define and evaluate a physical semantic join operator that leverages advantages of semantic data integration frameworks.

Chapter 6 delves into query processing over knowledge graphs. Querying is a crucial interaction point where the market value of a knowledge graph is defined by its capabilities including analytics, and analytics, in turn, largely rely on query processing. We show how querying of large-scale semi-structured knowledge graphs can be improved using novel logical integration and query optimization approaches, i.e., we define a multi-way join operator for semantic query processing and new query planning techniques that incorporate this operator.

Finally, Chapter 7 concludes the thesis with the directions of future work. We once more look through the research questions and provide answers using the obtained results.

CHAPTER 2

Background

In this chapter, we present basic concepts that serve as foundations of the research conducted in this thesis. Firstly, in Section 2.1 we introduce a stack of semantic technologies that back up the Linked Data concept. As the thesis discusses Enterprise Knowledge Graphs in Section 2.2, we cover the general idea of knowledge graphs, describe their characteristics and diversity. Then, Section 2.3 provides more details on semantic query processing with SPARQL and includes typical query optimizations and join techniques. As to data integration which is a necessary component of enterprise data management systems, we give an overview of the Data Integration System concept in Section 2.4.

2.1 Semantic Web & Linked Data

Historically, the Web is a collection of interlinked HTML documents accessible via the HyperText Transfer Protocol (HTTP). The *Web of Documents* is mostly human-centered and not suitable for intelligent machine agents. The concept of the *Semantic Web* (or *Web of Data*) proposed by Berners-Lee [1] aims at adding *meaning* to the information stored in the Web thus allowing machine agents to understand the context of data. Semantic Web therefore contains machine-readable descriptions of resources (things) that can be real-world entities such as "laptop" as well as abstract concepts such as "friendship". Having an explicitly declared meaning of each resource facilitates intelligent context-based search in contrast to common full-text search. For example, context-based semantic search is able to correctly return "Mona Lisa" as the answer of the query "The most famous painting of Leonardo Da Vinci in the Louvre Museum" whereas traditional full-text engines would return lists of documents related to the Louvre and Leonardo Da Vinci without a direct link to "Mona Lisa".

The stack of technologies the Semantic Web is based on is presented in Fig. 2.1. Each resource, e.g., document, entity, concept, has a unique identifier standardized by W3C as a Uniform Resource Identifier (URI) [5]. URI has to be encoded using Unicode characters and dereferencable via HTTP, e.g., the URI for the `Mona Lisa` resource might look like `http://dbpedia.org/resource/Mona_Lisa`. Syntactically, the Semantic Web relies on Extensible Markup Language (XML) [6] and its descendants as Turtle [1], N-Triples [2], N3 [3], TriG [4]. The core of the stack is the Resource Description Framework (RDF) [7] which we cover in details in Section 2.1.1. RDF employs a triple model, i.e., `subject predicate object` statements, in order to provide a formal resource description. Formal semantics

[1] Turtle Specification `https://www.w3.org/TR/turtle/`

[2] N-Triples Specification `https://www.w3.org/TR/n-triples/`

[3] N3 Specification `https://www.w3.org/TeamSubmission/n3/`

[4] TriG Specification `https://www.w3.org/TR/trig/`

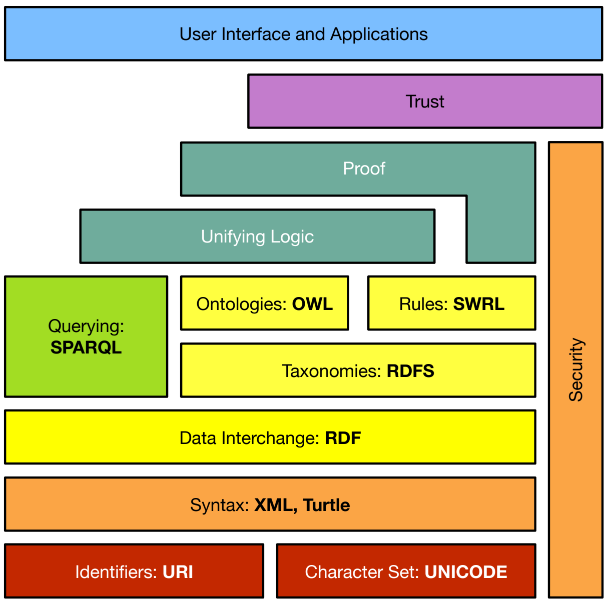


Figure 2.1: **Semantic Web technology stack.** Whereas lower levels comprise syntactic part of the concept, upper parts gravitate towards semantics and logical part of the concept.

in RDF is attained via a standardized *vocabulary*, i.e., a collection of predicates with the determined meaning. The next two levels of the stack enrich descriptive capabilities of RDF. Section 2.1.2 discusses RDF Schema (RDFS) that introduces semantics for hierarchical structures, and Web Ontology Language (OWL) that provides even more powerful capabilities for formalizing complex ontologies. Semantic Web Rules Language (SWRL) standard defines formal semantics for logical rules, i.e., causal relationships among resources and properties, that add to the overall expressivity of the RDF data model. SPARQL Protocol and RDF Query Language is a standardized language for semantic querying RDF data and knowledge graphs. Graph-traversing techniques behind SPARQL make it resemble a NoSQL approach. Since querying is utterly important for the research problem of this thesis we dedicate Section 2.3 for a closer look at SPARQL. Formal semantics of RDF stems from the Description Logics (DL) which is in turn based on the First-order Logic (FOL). The Unifying Logic and Proof layers of the stack utilize formal semantics to enable machine reasoning and therefore automated intelligent agents. However, at the moment there is no standardized means for Unifying Logic and Proof layers. Nevertheless, security mechanisms such as cryptography, TLS, HTTPS accompany the above-mentioned layers and constitute a data protection layer of the stack. Eventually, with the adoption of automated semantic verification and provenance the overall trust in the Web increases, and the knowledge contained in the Semantic Web can be used by applications and consumers.

Linked Data denotes a set of best practices that implement the vision of the Semantic Web in actual use-cases. Tim Berners-Lee defined [8] four principles Linked Data should adhere to:

- URIs are used as names of the things;
- HTTP URIs should be used for dereference so that people can look up those names;
- When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL);
- Include links to other URIs so that people can discover more things.

The message behind these principles is to promote openness and interlinking of information in both human and machine-readable ways. This message inspired data providers to publish their datasets under open licences following the above recommendations which in turn led to emergence of the *Linked Open Data Cloud* (LOD Cloud) [9, 10], a network of interconnected semantic datasets.

As of 2018, LOD Cloud consists of thousands of datasets and billions of RDF-encoded facts thanks to both community and enterprises efforts. Among the largest community-supported knowledge bases are DBpedia [11] – a machine-readable version of Wikipedia [5]; Wikidata [12] – an envisioned uniform source for Wikipedia articles; LinkedGeoData [13] – a collection of geospatial data from OpenStreetMap [6]. As to enterprise data, corporations contribute to the LOD Cloud with their own sources. For instance, *datahub.io*[7] contains hundreds of datasets published by companies and state institutions.

LOD Cloud datasets are encouraged to share the same schema, i.e., *vocabulary*. Numerical, statistical, observational data employs RDF Data Cube [8] vocabulary. Categorical and hierarchical structures are often modelled by Simple Knowledge Organization System (SKOS) [9] or Dublin Core (DC) [10] vocabularies. Friend of a Friend (FOAF) [11] is the most used vocabulary for social information and linking people.

2.1.1 The Resource Description Framework (RDF)

The Resource Description Framework (RDF) is a semantic graph-based data model tailored for representing semi-structured data, i.e., real-world or abstract concepts, on the Web. RDF is a W3C standard [7] that specifies architecture, syntax, semantics, and a basic vocabulary, that is, RDF Schema (RDFS) [14].

The main building block of RDF is a *triple*. RDF triple is a positive statement encoded in the form of `subject predicate object` where:

- Subject denotes a resource to which predicate and object belong; only URIs or blank nodes can be subjects in RDF;
- Predicate denotes a trait of the subject, i.e., a relationship between the subject and object; only URIs can be predicates in RDF;
- Object specifies predicate with a particular value; URIs, blank nodes or string literals can represent a value of the predicate.

Fig. 2.2 illustrates an example of an RDF triple with the statement *Falcon 9 is a rocket*. The subject of the triple is a resource URI `ex:Falcon9`, the predicate is a URI `rdf:type` that defines a

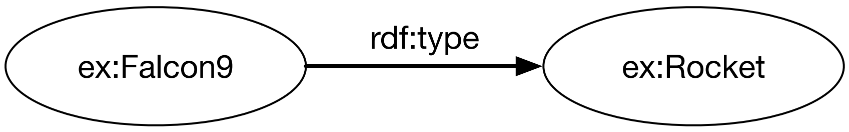


Figure 2.2: **An example RDF triple.** `ex:Falcon9` is a subject node, `ex:Rocket` is an object node. Two nodes are connected via the `rdf:type` predicate.

[5]http://www.wikipedia.org
[6]https://www.openstreetmap.org/
[7]http://datahub.io/
[8]https://www.w3.org/TR/vocab-data-cube/
[9]https://www.w3.org/2009/08/skos-reference/skos.html
[10]http://dublincore.org/documents/dc-rdf/
[11]http://xmlns.com/foaf/spec/

semantic relation *is a*, and the object is a resource URI `ex:Rocket`. Prefixes `ex:` and `rdf:` abbreviate full namespaces of those URIs, so that `ex:Falcon9` is a shortened version of the full URI `http://www.example.org/Falcon9`. As RDF is a graph-based model, every statement can also be presented as a directed graph where subject and object are nodes, and a predicate is an edge between two vertices. Formally, RDF triple is defined as follows [7, 15]:

RDF Triple

Definition 2.1.1 *Let $\mathcal{U}, \mathcal{B}, \mathcal{L}$ be disjoint infinite sets of URIs, blank nodes, and literals, respectively. A tuple $(s, p, o) \in (\mathcal{U} \cup \mathcal{B}) \times (\mathcal{U}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ is denominated as an RDF triple, where s is called the subject, p the predicate, and o the object. (s,p,o) is a generalized RDF triple when $(s, p, o) \in (\mathcal{U} \cup \mathcal{B}) \times (\mathcal{U}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$. An element $\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$ is called an RDF term.*

Hence, the triple in Fig. 2.2 is formally defined as `(ex:Falcon9, rdf:type, ex:Rocket)`. In the previous example, we covered so far only URIs. Literals and blank nodes can be components of an RDF triple as well. A literal is a string or numerical constant that can be put as a value of a predicate. Literals can have a datatype prescribed in the XML Schema standard [16] or a language tag according to the BCP 47 [17], e.g., `"0.5"^^<http://www.w3.org/2001/XMLSchema#float>` defines a literal of a float datatype with 0.5 as a value. Similarly, `"Germany"@en` and `"Deutschland"@de` define string constants in English and German language, respectively. A blank node is a resource without URI that is accessible only internally within an identified resource that wraps this blank node with some predicate. Blank nodes are often used to model containers, lists and collections. For instance, describing the following statement *SpaceX owns a hangar with six rockets* in RDF, *a hangar* can be a blank node that contains *rocket* instances. Semantically, blank nodes resemble an existential quantifier.

As a semi-structured model, RDF provides a semantic basis for logical constructs such as existential and universal quantifiers. Following a standardized set of rules, RDF enables logical reasoning and inference of new statements. Therefore, a data model is by default incomplete and assumes an undefined truth value of each statement that is not yet explicitly described by RDF triples. The latter feature is a characteristic of the *Open World Assumption (OWA)* modeling paradigm. In the Semantic Web, OWA allows data models, e.g., vocabularies and ontologies, to be enriched over time with new statements without the need to align all existing triples to the updated version. On the other hand, at the large scale of Big Data and Big Knowledge with thousands of agents OWA might lead to schema inconsistency and logical errors. In contrast, traditional relational data models are rigidly structured thus adhering to the *Closed World Assumption (CWA)* paradigm where every statement is assumed false if not present in the current schema. That is, adding a new predicate to a relational model would result in assigning a certain value, e.g., null, none, zero, or empty string, for all existing records.

2.1.2 Extended Expressivity with RDFS and OWL

RDF itself standardizes a triple-based graph model that supports basic descriptions like that *something* has some *type*. However, RDF is limited and does not provide semantic means to define classes, i.e., groups of resources that share the same attributes, nor customize properties with their allowed values.

The RDF Schema (RDFS) [14] extends the RDF model with new predicates to formalize *classes* via the predicate `rdfs:Class`, literals via `rdfs:Literal`, enabling hierarchical structures, e.g., classes hierarchies via the predicate `rdfs:subClassOf`, properties hierarchies via `rdfs:subPropertyOf`. Additionally, RDFS enables definitions of *ranges* and *domains* for properties. A class of the object of a predicate is called *range* and modeled via the `rdfs:range` predicate. Similarly, a class of the subject of a predicate is called *domain* and described by the `rdfs:domain` predicate. Moreover, to support human-readability RDFS introduces predicates such as `rdfs:label`, `rdfs:comment`, `rdfs:seeAlso`, `rdfs:isDefinedBy` that contain human-friendly natural language text annotations. To sum up, the following example in Listing 2.1 expressed in the Turtle syntax illustrates the usage of RDFS predicates applied to the `ex:Rocket` concept from Fig. 2.2:

```
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex:     <http://www.example.org/> .

ex:Rocket       rdfs:subClassOf     ex:Vehicle .
ex:Spaceship    rdfs:subClassOf     ex:Payload .
ex:Cargo        rdfs:subClassOf     ex:Payload .
ex:transports   rdfs:domain         ex:Rocket .
ex:transports   rdfs:range          ex:Payload .
ex:transports   rdfs:subPropertyOf  ex:moves .
```

Listing 2.1: RDFS example of six triples which define subclass axioms for classes `Rocket`, `Spaceship`, and `Cargo`, and axioms describing range, domain, and hierarchical relation of a property `transports`.

With RDFS we are able to express that every `Rocket` is a `Vehicle`, `Rocket` transports `Payload` into orbit. `Payload` might vary from manned `Spaceship` to automatic `Cargo`. Transporting payload into orbit is a specialized case of moving an object, therefore `transports` is a subproperty of `moves`. Semantics behind RDFS enables basic inference. For instance, given a new triple `ex:Falcon9 ex:transports ex:Dragon`, a reasoner can deduce based on the range and domain of the predicate that `ex:Falcon9` belongs to the `ex:Rocket` class whereas `ex:Dragon` is a `ex:Payload`.

RDFS is often employed as a basic *vocabulary* for more complex *ontologies*. While both vocabularies and ontologies are defined in RDF, vocabularies, e.g., RDFS, SKOS, FOAF, are less complex and provide a set of predicates with a pre-defined meaning. In contrast, ontologies, as defined by Gruber [18], are "explicit specification of a conceptualization", and thus exhibit more sophisticated nature with numerous classes, properties, and restrictions on those properties, e.g., Gene Ontology [19, 20], SNOMED-CT [21].

The Web Ontology Language (OWL) [22] is a complex extension of RDFS leveraged by the above-mentioned ontologies. OWL introduces new logical constructs and formal semantics at each level of the RDF model whereas some of them are complex enough so that models become *undecidable*, i.e., a logical reasoning mechanism is not able to produce complete results. A subset of new constructs is:

- Constants: top (universal) class `owl:Thing`, empty class `owl:Nothing`, top and empty properties;
- Class level: boolean connectivity, e.g., `owl:unionOf`, disjointness (`owl:disjointWith`) and equivalence (`owl:equivalentClass`) of classes;
- Instance level: equivalence of instances (`owl:sameAs`) or difference (`owl:differentFrom`);
- Predicate level: object value assertions, e.g., a URI for `owl:ObjectProperty`, a literal for `owl:DatatypeProperty`, cardinality restrictions, universal and existential quantifiers, complex assertions, e.g., if a `owl:FunctionalProperty` can have only one value.

```
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex:      <http://www.example.org/> .

ex:Rocket        owl:disjointWith    ex:Animal .
ex:transports    rdf:type            owl:ObjectProperty .
ex:transports    rdf:type            owl:AsymmetricProperty .
```

Listing 2.2: OWL example. Classes `Rocket` and `Animal` are annotated as disjoint, a property `transports` is annotated as an asymmetric object property.

Listing 2.2 illustrates how OWL enriches the model with additional semantics. It is rather obvious that no rocket can be an animal at the same time, i.e., two classes are disjoint. This statement is expressed in OWL via the `owl:disjointWith` predicate. Then, in case another agent inserts a statement `ex:Falcon9 rdf:type ex:Animal` the model becomes incorrect because of a probable mistake - falcons are animals but not rockets. Secondly, the predicate `ex:transports` is annotated as an `owl:ObjectProperty` which means that the value of this predicate is always an instance of some `rdfs:Class`. Moreover, the statement infers that `ex:Payload` (as a *range* of the predicate) is an `rdfs:Class` which was not explicitly defined in Listing 2.1. Another restriction does not allow payload to transport rockets, i.e., `ex:transports` is asymmetric with fixed range and domain classes, hence the predicate is annotated as `owl:AsymmetricProperty`.

2.2 Knowledge Graphs

Knowledge management approaches date back to the early efforts on computer systems aimed at implementing various features of artificial intelligence (AI). *Expert systems* were the first working attempt [23] on symbolic reasoning. Expert systems usually consist of two components: a knowledge base that stores facts and rules, and an inference engine that performs actual reasoning. Often LISP or Prolog machines, the expressivity and usability of expert systems was limited so that in early 1990s they became highly specialized and tailored for a certain domain, and the concept of AI based on expert systems lost its ground. Probabilistic approaches, or *Bayesian networks*, represent another branch [24] in knowledge management and inference. Bayesian networks rely on statistics and conditional (in)dependencies employing Markov models and Monte Carlo methods. However, probabilistic reasoning lacks explicit semantics and schema hiding them inherently in conditional computations.

A *knowledge graph* is a novel knowledge organization paradigm whereas the term has been coined by Google since 2012 [25]. The concept received a significant traction in the community so that the term was used to refer to private and public graph-based collections of knowledge, e.g., DBpedia [11], Wikidata [12], YAGO [26] are prominent public general-purpose knowledge graphs. Yahoo [27], Microsoft [12], Facebook [13] developed their own knowledge graphs similar to Google's. As there is no community agreement upon a formal definition of knowledge graphs, Paulheim [28] provides a description of typical characteristics of knowledge graphs:

[12] http://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing/
[13] http://www.facebook.com/notes/facebook-engineering/under-the-hood-the-entities-graph/10151490531588920

Knowledge graph (Paulheim [28])

Definition 2.2.1 *A knowledge graph:*

1. *mainly describes real world entities and their interrelations, organized in a graph;*
2. *defines possible classes and relations of entities in a schema;*
3. *allows for potentially interrelating arbitrary entities with each other;*
4. *covers various topical domains.*

As follows from Definition 2.2.1, knowledge graphs may employ different knowledge representation formalisms including abstract modeling languages and probabilistic mechanisms not limiting itself to purely RDF. As to domains, knowledge graphs might be applicable in manifold ways, e.g., scientific knowledge graphs [29], biomedical knowledge graphs [30], and many are yet to arrive. In this thesis, we focus on knowledge graphs based on the Semantic Web stack, i.e., RDF graphs with explicit schema provided by ontologies. RDF graphs are defined as follows:

RDF Graph [15]

Definition 2.2.2 *Let $\mathcal{U}, \mathcal{B}, \mathcal{L}$ be disjoint infinite sets of URIs, blank nodes, and literals, respectively. An RDF graph G is a directed labeled multigraph $G = (\mathcal{N}, E, \Sigma, \mathcal{L})$ where:*

- $\mathcal{N} \subset (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ *a finite set of RDF terms that correspond to nodes*
- $E \subseteq \mathcal{N} \times \mathcal{N}$ *a finite set of edges that connect RDF terms*
- $\Sigma \subset \mathcal{U}$ *a set of labels uniquely identified with URIs*
- $\mathcal{L} : E \rightarrow 2^{\Sigma}$ *that maps edges to sets of labels*

Fig. 2.3 illustrates an example RDF graph that describes products and companies owned by *Elon Musk*. The graph contains all RDF terms, namely URIs (e.g., `ex:Person` as a class, `ex:Elon_Musk` as an instance of this class), literals (e.g., a full title of SpaceX expressed in English `"Space Exploration Technologies Corp."@en`), and a blank node that encapsulates a stock price numerical value `272.49` and its currency as a URI `ex:Euro`.

2.3 Query Processing with SPARQL

SPARQL is a recursive acronym SPARQL Protocol and RDF Query Language. SPARQL is recommended by W3C [31] for querying RDF datasets by means of a graph pattern matching approach. As querying is an important part of the thesis we take a closer look at the basics of SPARQL in Section 2.3.1, how to access RDF datasets via SPARQL in Section 2.3.2, and foundations of query processing in Section 2.3.3.

2.3.1 SPARQL as a Query Language

The main building block of a SPARQL query is a *triple pattern*. Similar to an RDF triple, a triple pattern consists of placeholders, i.e., variables, at positions of a subject, predicate, or object (or at all of them simultaneously). During the evaluation of a query patterns are matched against the target RDF dataset so that variables turn into RDF terms that constitute a query solution mapping. *Basic Graph Pattern (BGP)* denotes a conjunction of a set of triple patterns.

Triple Pattern, Basic Graph Pattern [15]

Definition 2.3.1 *Let $\mathcal{U}, \mathcal{B}, \mathcal{L}$ be disjoint infinite sets of URIs, blank nodes, and literals, respectively. Let V be a set of variables such that $V \cap (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}) = \emptyset$. A triple pattern tp is member of the set $(\mathcal{U} \cup V) \times (\mathcal{U} \cup V) \times (\mathcal{U} \cup \mathcal{L} \cup V)$. Let $tp_1, tp_2, \ldots, tp_n$ be triple patterns. A Basic Graph Pattern (BGP) B is the conjunction of triple patterns, i.e., $B = tp_1$ AND tp_2 AND $\ldots$ AND tp_n.*

SPARQL allows conjunctions of triple patterns to be extended with filters (`FILTER`), optional patterns (`OPTIONAL`), logical operators (`UNION` and `AND`), aggregate functions. Furthermore, SPARQL defines four query forms: `SELECT`, `ASK`, `CONSTRUCT`, and `DESCRIBE`. `SELECT` query is the most used form as it returns a set of bound variables, i.e., RDF terms from the graph that satisfy the given BGP. `ASK` query return a boolean value `TRUE` if the given BGP has solutions in the target RDF dataset or `FALSE` otherwise. `CONSTRUCT` query returns an RDF graph build according to the BGP. `DESCRIBE` query result is an RDF graph with relevant information about the URIs in a BGP. The definition is intentionally left vague in order to let query engines determine this *relevant information*. In this thesis, we focus on querying RDF datasets, i.e., RDF knowledge graphs, with the `SELECT` query. Formally, SPARQL expression and the `SELECT` query are defined as:

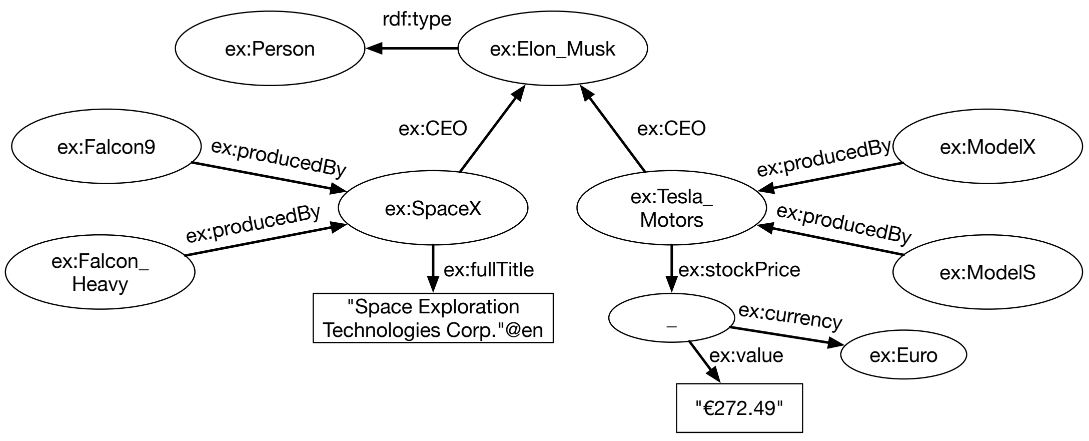


Figure 2.3: **RDF graph example.** The graph consists of classes, individuals, literals, and blank nodes.

SPARQL Expression and SELECT Query [15]

Definition 2.3.2 *Let V be a set of variables. A SPARQL expression is built recursively as follows.*

1. *A tuple from* $(\mathcal{U} \cup V) \times (\mathcal{U} \cup V) \times (\mathcal{U} \cup \mathcal{L} \cup V)$ *is a triple pattern.*
2. *If* Q_1 *and* Q_2 *are graph patterns, then expressions* $(Q_1$ *AND* $Q_2)$, $(Q_1$ *UNION* $Q_2)$, $(Q_1$ *OPT* $Q_2)$ *are graph patterns and SPARQL expressions, i.e., conjunctive graph pattern, union graph pattern, and optional graph pattern, respectively.*
3. *If Q is a SPARQL expression and R is a SPARQL filter condition, then (Q FILTER R) is a SPARQL expression, i.e., filter graph pattern.*

If Q is a SPARQL expression and $S \subset V$ *a finite set of variables, then SPARQL SELECT query is an expression of the form* $SELECT_S(Q)$.

Listing 2.3 illustrates an example of a SPARQL `SELECT` query that returns *products that are produced by companies Elon Musk has a CEO position at* evaluated against the RDF graph in Fig. 2.3. BGP of the query contains a conjunction of two triple patterns tp_1 and tp_2 that share the same variable `?company`.

```
PREFIX ex:     <http://www.example.org/> .

SELECT ?product WHERE {
?product         ex:producedBy    ?company          .    #tp1
?company         ex:CEO           ex:Elon_Musk      . }  #tp2
```

Listing 2.3: SPARQL SELECT query example that extracts all products produced by a company which has Elon Musk as a CEO.

SPARQL Semantics

Each possible answer in the given RDF dataset for each triple pattern belongs to *SPARQL mappings*. Mappings have to be identified during the evaluation of the query.

SPARQL Mapping [15]

Definition 2.3.3 *A mapping is a partial function* $\mu : V \rightarrow \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$ *from a subset of variables to RDF terms. The domain of the mapping* μ, $dom(\mu)$, *is the subset of variables V for which* μ *is defined. Two mappings* μ_1, μ_2 *are compatible, i.e.,* $\mu_1 \sim \mu_2$, *if* $\forall x \in dom(\mu_1) \cap dom(\mu_2) : \mu_1(x) = \mu_2(x)$. *vars(tp) is a function that returns all variables in triple pattern tp whereas* $\mu(tp)$ *is the triple pattern obtained when replacing all* $x \in dom(\mu) \cap vars(tp)$ *by* $\mu(x)$.

Operations on SPARQL mappings sets are governed by the SPARQL algebra.

SPARQL Algebra [15]

Definition 2.3.4 *Let $\Omega, \Omega_l, \Omega_r$ be SPARQL mapping sets, R a filter condition, and S is a finite set of variables. SPARQL algebra defines the following operations:*

$$\begin{aligned}
\Omega_l \bowtie \Omega_r &:= \{\mu_l \cup \mu_r \mid \mu_l \in \Omega_l, \mu_r \in \Omega_r : \mu_l \sim \mu_r\}\\
\Omega_l \cup \Omega_r &:= \{\mu \mid \mu \in \Omega_l \vee \mu \in \Omega_r\}\\
\Omega_l \setminus \Omega_r &:= \{\mu_l \in \Omega_l \mid \forall \mu_r \in \Omega_r : \mu_l \nsim \mu_r\}\\
\Omega_l ⟕ \Omega_r &:= \{(\Omega_l \bowtie \Omega_r) \cup (\Omega_l \setminus \Omega_r)\}\\
\pi_S(\Omega) &:= \{\mu_1 \mid \exists \mu_2 : \mu_1 \cup \mu_2 \in \Omega \wedge dom(\mu_1) \subseteq S \wedge dom(\mu_2) \cap S = \emptyset\}\\
\sigma_R(\Omega) &:= \{\mu \in \Omega \mid \mu \models R\}
\end{aligned}$$

Where $\models$ tests that a mapping μ satisfies the filter condition R.

The rules of re-writing a SPARQL query with its expressions into SPARQL algebra are specified by SPARQL *set semantics*, that is, a set of mappings is expected as the output of query evaluation.

SPARQL Set Semantics [15]

Definition 2.3.5 *Let D be an RDF dataset, tp a triple pattern, Q, Q_1, Q_2 SPARQL expressions, R a filter condition, S a finite set of variables. Let $[[\cdot]]_D$ be a function that translates SPARQL expressions into SPARQL algebra operators as follows:*

$$\begin{aligned}
[[tp]]_D &:= \{\mu \mid dom(\mu) = vars(tp) \wedge \mu(tp) \in D\}\\
[[Q_1\ AND\ Q_2]]_D &:= [[Q_1]]_D \bowtie [[Q_2]]_D\\
[[Q_1\ OPT\ Q_2]]_D &:= [[Q_1]]_D ⟕ [[Q_2]]_D\\
[[Q_1\ UNION\ Q_2]]_D &:= [[Q_1]]_D \cup [[Q_2]]_D\\
[[Q\ FILTER\ R]]_D &:= \sigma_R([[Q]]_D)\\
[[SELECT_S(Q)]]_D &:= \pi_S([[Q]]_D)
\end{aligned}$$

Where $\models$ tests that a mapping μ satisfies the filter condition R.

Therefore, applying the above definitions to the example query in Listing 2.3, the engine would evaluate the following expression $Q = tp_1$ `AND` tp_2, where:
tp_1 = `(?product, ex:producedBy, ?company)`
tp_2 = `(?company, ex:CEO, ex:Elon_Musk)`
According to the SPARQL semantics, the evaluation of Q over RDF dataset D in Fig. 2.3 corresponds to: $[[Q]]_D = [[tp_1]]_D \bowtie [[tp_2]]_D$. Then, the mapping sets Ω_1 and Ω_2, i.e., instantiations of tp_1 and tp_2, respectively, are:

$\Omega_1 = \{$

$\mu_1 = \{$`product` $\rightarrow$ `ex:Falcon9`, `company` $\rightarrow$ `ex:SpaceX`$\}$

$\mu_2 = \{$`product` $\rightarrow$ `ex:Falcon_Heavy`, `company` $\rightarrow$ `ex:SpaceX`$\}$

$\mu_3 = \{$`product` $\rightarrow$ `ex:ModelX`, `company` $\rightarrow$ `ex:Tesla_Motors`$\}$

$\mu_4 = \{$`product` $\rightarrow$ `ex:ModelS`, `company` $\rightarrow$ `ex:Tesla_Motors`$\}\}$

$\Omega_2 = \{$

$\mu_5 = \{$`company` $\rightarrow$ `ex:SpaceX`$\}$

$\mu_6 = \{$`company` $\rightarrow$ `ex:Tesla_Motors`$\}\}$

Performing a join operation $\Omega_1 \bowtie \Omega_2$, compatible mappings from the two sets should be merged, i.e., $\mu_l \cup \mu_r \mid \mu_l \in \Omega_1, \mu_r \in \Omega_2 : \mu_l \sim \mu_r$. $\mu_1 \sim \mu_5$, and $\mu_2 \sim \mu_5$ are compatible as they share the same variable `company` $\rightarrow$ `ex:SpaceX` where the projected variable `product` has RDF terms `ex:Falcon9` and `ex:Falcon_Heavy` . Similarly, $\mu_3 \sim \mu_6$, and $\mu_4 \sim \mu_6$ are compatible since they share the same `company` $\rightarrow$ `ex:Tesla_Motors` where the projected variable `product` is instantiated with `ex:ModelX` and `ex:ModelS`. Therefore, the final solution of Q against D is:

$[[Q]]_D = \{$

$\{$`product` $\rightarrow$ `ex:Falcon9`$\}$

$\{$`product` $\rightarrow$ `ex:Falcon_Heavy`$\}$

$\{$`product` $\rightarrow$ `ex:ModelX`$\}$

$\{$`product` $\rightarrow$ `ex:ModelS`$\}\}$

2.3.2 SPARQL Access Interfaces

Raw RDF dumps do not have any default built-in query interfaces. In order to send SPARQL queries against an RDF dataset, such a dataset has to be deployed via a SPARQL access interface. If the dump is envisioned to be published on the Web or be a part of the LOD Cloud, then the interface should provide HTTP means as well. Access interfaces are often combined with a persistent RDF storage that transforms raw RDF dumps into inner representations and maintains indexes to speed up query processing. There exist two major and widely used SPARQL access interfaces: SPARQL endpoints and Linked Data Fragments. Querying approaches have to deal with a common trade-off between supported expressivity and performance. Higher expressivity allows engines to evaluate more complex queries and save network requests to the server that hosts the access interface. On the other hand, complex queries tend to consume more time during the execution which decreases the performance. SPARQL endpoints and Linked Data Fragments appear on the opposite sides of the *expressivity–performance* trade-off, i.e., SPARQL endpoints are able to execute complex queries whereas Linked Data Fragments perform faster on less complex queries in exchange for higher network costs.

SPARQL Endpoint

Definition 2.3.6 *A Web service that provides HTTP-based means for querying RDF datasets via the SPARQL protocol is denominated as a SPARQL endpoint. The endpoint implements a function f that maps from a query Q to solution mappings μ:*

$$f : Q \to \mu$$

SPARQL endpoints receive queries from software or human agents and return solution mappings as specified in the SPARQL standard and Section 2.3.1. Endpoints must be able to evaluate all syntactically correct queries that adhere to the standard, hence SPARQL endpoints retain high expressivity of the original standard. Public endpoints are exposed to millions of queries weekly and monthly [32, 33], thus query complexity imposes significant performance issues. Due to those factors public endpoints exhibit low availability as reported in [34, 35]. It is therefore a common practice for publicly accessible endpoints to place restrictions on the estimated execution time, or total execution time, or number of returned solution mappings, e.g., for the DBpedia endpoint [14] the limit on the number of answers is 10000 triples. Being a server-centric approach, SPARQL endpoints perform query evaluation on its own side, i.e., server-side. Recent advances [36, 37] in hybridization of query execution between server and client demonstrate a possibility to improve query performance.

Linked Data Fragments (LDF) is a formal concept introduced by Verborgh et al. [38] that shifts a significant part of query processing to the client side. The concept envisions Linked Data interfaces available on the Web that evaluate rather simple triple patterns of less expressive SPARQL subset thus increasing query performance.

Linked Data Fragment (LDF) [38]

Definition 2.3.7 *Let $G \subseteq T$ be a finite set of blank-node-free RDF triples. A Linked Data Fragment (LDF) of G is a tuple $f = \langle u, s, \Gamma, M, C \rangle$ where:*

- *u is a URI of the source from which f can be retrieved;*
- *s is a selector function;*
- *Γ is a set of RDF triples that is the result of applying s to G, i.e., $\Gamma = s(G)$;*
- *M is a set of metadata RDF triples;*
- *C is a finite set of hypermedia controls.*

Due to the size of obtained results fragments are split into pages with references to the previous and next pages. A set of all possible fragments for a given source is denominated as an *LDF collection*. Triple

[14] http://dbpedia.org/sparql

Pattern Fragments (TPF) is an implementation of the Linked Data Fragments approach. TPF provides a uniform client-server architecture.

Triple Pattern Fragment (TPF) [38]

Definition 2.3.8 *Given a control c, a c-specific LDF collection F is called a Triple Pattern Fragment collection if, for every possible triple pattern tp, there exists one LDF* $\langle u, s, \Gamma, M, C\rangle \in F$*, referred to as a Triple Pattern Fragment, that has the following properties:*

- *s is a triple-pattern-based selector function for tp;*
- *there exists a triple* $\langle u, \textit{void:triples}, \textit{cnt}\rangle \in M$ *where cnt is an estimate cardinality of* Γ*;*
- $c \in C$.

The TPF server hosts RDF datasets and is best-optimized for single triple pattern queries.

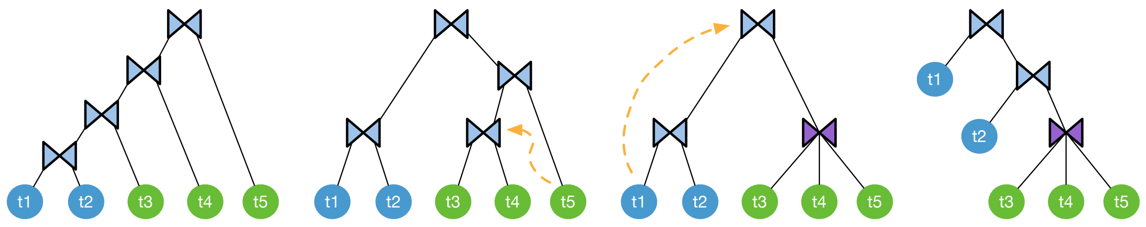
TPF Query Semantics [38]

Definition 2.3.9 *Let* $G \subseteq T$ *be a finite set of blank-node-free RDF triples, and let F be some TPF collection over G. The evaluation of a SPARQL graph pattern P over F, denoted by* $[[P]]_F$*, is defined by* $[[P]]_F = [[P]]_G$.

In contrast to SPARQL endpoints, the TPF server returns actual RDF triples instead of solution mappings. The simplicity promotes higher availability of TPF servers on the Web. Moreover, the server does not limit the amount of its output to some constant value, i.e., all relevant *fragment pages* are returned. The evaluation of a SPARQL query produces a *fragment* split into *fragment pages*. Each fragment page consists of RDF triples, metadata (e.g., total number of query answers), and link to the next page. When posing more complex queries (of more than one triple pattern), the TPF client has to perform query decomposition and planning, implement SPARQL operators, e.g., joins, and process the fragments obtained from the server. Multiple requests for fragments increase network traffic and slow down the overall processing. That is, the burden of producing an answer is moved to the client side.

2.3.3 Query Processing

This paragraph provides a view on the basics of query processing with less emphasis on the RDF data model and SPARQL standard. Having translated a SPARQL query into algebraic expressions, the evaluation process that takes place inside a query engine resembles the process well-studied in the database community [39]. As data retrieval per se is a database task, a query engine has to optimize what happens before, during, and after the retrieval. Conceptually, the process is split into two stages: *i)* An optimizer devises a query plan that specifies physical operators to be used and their application sequence that comprises a tree, i.e., a query tree plan, where its root corresponds to a query answer, internal nodes



(a) Left-linear plan (b) Bushy tree plan (c) Multi-way join plan (d) Hybrid plan

Figure 2.4: **Query tree plan shapes and adaptivity.** (a) An initial left-linear plan. (b) The bushy plan after applying inter-operator adaptivity technique where the left branch in blocking and the right branch is non-blocking. (c) The plan with a multi-way join in the right branch as a result of intra-operator adaptivity. (d) The eventual hybrid plan.

to physical operators, and leaves to data retrieval. *ii)* The engine executes the plan and implements the operators in order to obtain query answers.

Query optimization is proved [40, 41] to be an NP-complete problem. Therefore, optimizers employ various heuristics and estimations to obtain an effective plan. Cost-based estimations often rely on *cardinality* of intermediate results produced by input operators, e.g., applied to SPARQL queries, each triple pattern has its own number of intermediate results and thus its own cardinality. Operators that have low cardinality demonstrate *high selectivity*, and, vice versa, *low selective* operators have lots of intermediate results. The cost value depends on overall cardinality, i.e., a combination of operators that produces less intermediate results has lower cost and is therefore a preferred execution option. Cardinality information is often taken from the dataset statistics. The drawbacks of cardinality-based cost models become significant in environments where statistics is not available at all or permanently changes, e.g., in federated setups remote sources do not provide any metadata on their contents, nor in dynamically updating datasets a reliable estimate can be computed.

Query tree plan is another subject for optimization. Optimizers usually aim to produce an efficient tree of join operators that is especially important in the case of complex queries, e.g., multiple triple patterns in SPARQL queries. Known to be a search problem, the size of the search space varies depending on the tree shape. For instance, Fig. 2.4 illustrates some of the shapes, e.g., a left-linear shape in Fig. 2.4(a), a bushy shape in Fig. 2.4(b),2.4(c), and their permutation in Fig. 2.4(d). As to join operators that are nodes in plan trees, the largest set of most applied operators consists of binary join operators, i.e., that perform a join of two inputs. In the relational database community, but not in SPARQL, multi-way joins, i.e., the operators that join multiple inputs simultaneously, gained a momentum, and plans in Fig. 2.4(c),2.4(d) show examples of three-way join operators.

Analyzing search strategies, three different approaches can be found in the database state-of-the-art [41], i.e., heuristic search, exhaustive search, and randomized search. The time complexity of heuristic algorithms such as [42] is polynomial. Exhaustive search algorithms given a complex query are less effective, i.e., dynamic programming-based techniques [43] require exponential time. The performance of randomized search algorithms such as Two Phase Optimization (2PO) [44] may vary depending on the query use-case, i.e., they can produce good plans as well as computationally very expensive plans.

In query optimization, there exist two paradigms, i.e., static optimization and continuous optimization. *Optimize-then-Execute* is a static paradigm where the plan produced by an optimizer using some of the above-mentioned strategies is fixed. That is, once the plan is constructed the engine strictly follows the order. For instance, Fig. 2.4(a) represents a plan derived by an optimizer where binary join operators are to be executed in the specified sequence. However, fixed plans do not consider runtime conditions that might change over time, e.g., network delays, data availability and updates, poor complexity estimations.

Adaptive query processing [45] is a continuous optimization paradigm that allows a plan to be changed in the ad-hoc and on-the-fly manner in order to adjust to changing runtime environment. Adaptivity is supported by metadata and statistics gathered by the query engine during the execution of a plan. To illustrate how adaptive optimizers change the plan, consider Fig. 2.4. Let Fig. 2.4(a) be a plan derived by a static optimizer and this plan is identified to perform slower than expected. It is known that the leftmost join operator (that joins `t1` and `t2`) is blocking, i.e., no further intermediate results are pushed forward until all input tuples of the blocking operator are processed. Therefore, the optimizer applies an *inter-operator* adaptivity technique by re-arranging other join operators into a new branch (Fig. 2.4(b)). Inter-operator adaptivity is beneficial in the presence of blocking operators that impede processing of other operators that can be executed independently from the blocking ones. Furthermore, the optimizer is able to improve the new branch using *intra-operator* adaptivity technique by merging two binary join operators into one multi-way join operator as shown in Fig. 2.4(c). Intra-operator adaptivity is applicable to non-blocking operators that produce intermediate results over time and push them to the next stages once a tuple is ready. That is, non-blocking operators do not have to wait until all input tuples arrive. Finally, after introducing several changes and obtaining new statistics, the optimizer recognizes that the original branch can be enhanced by breaking down the blocking join operator into two joins as depicted in Fig. 2.4(d), i.e., applying inter-operator adaptivity once again. Now the eventual plan performs faster than expected compared to the initial version thanks to the adaptive approach.

2.4 Data Integration System

It is natural for enterprises to have dozens of data sources that are involved into relevant business processes. On the other hand, a variety of sources and data models hinders efforts towards a one logical representation of the whole enterprise data. The task of data integration implies combination of sources with heterogeneous schemas or models into a unified view suitable for, e.g., knowledge graph construction. This section presents a concept of a *Data Integration System (IS)* and its components. Then, we give an overview of two classic approaches for building a data integration system in Sections 2.4.1 and 2.4.2.

Data Integration System [46]

Definition 2.4.1 *A data integration system $\mathcal{IS}$ is defined as a tuple $\langle O, S, M \rangle$, where*

- *O is the **global schema** (ontology), expressed in a language $\mathcal{L}_O$ over an alphabet $\mathcal{A}_O$. The alphabet $\mathcal{A}_O$ consists of symbols for each element in O.*
- *S is the **source schema**, expressed in a language $\mathcal{L}_S$ over an alphabet $\mathcal{A}_S$. The alphabet $\mathcal{A}_S$ contains sybmbols for each element of the sources.*
- *M is the **mapping** between O and S that is represented as assertions:*

$$q_S \rightarrow q_O$$
$$q_O \rightarrow q_S$$

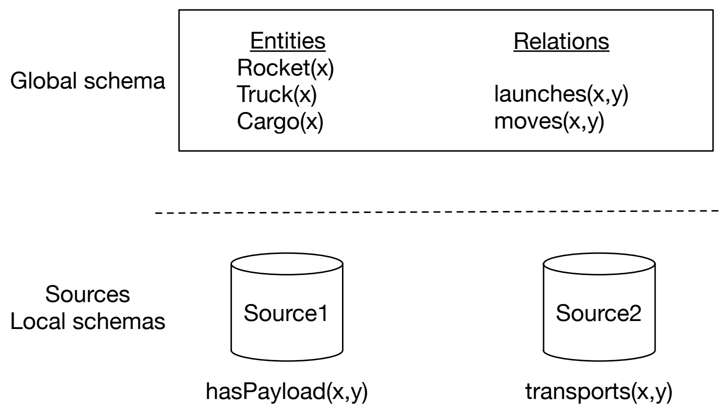


Figure 2.5: **An example of a data integration task.** Two sources S_1 and S_2 that contain local schemas have to be aligned with the global schema. The global schema defines three entities and two relations between them.

where q_S and q_O are two queries of the same arity, q_S is expressed in the source schema and q_O – in the global schema. The assertions imply correspondence between global and source concepts.

Aiming to achieve a unified view on different data sources, the global schema O is the actual provider of this unified view. The global schema supplies a set of concepts able to embrace a variety of source models. It is a common practice [47, 48] to employ ontologies as a global schema. Applications of RDF and OWL ontologies in data integration established the *ontology-based* and *semantic* data integration research fields [49, 50]. Two main approaches in the literature [51, 52] for creating and maintaining mappings in query processing against a data integration system are *Local-as-View* (LAV) and *Global-as-View* (GAV).

We illustrate a typical data integration task in Fig. 2.5, i.e., two local sources S_1 and S_2 with their own schemas have to be aligned with the global schema O. The global schema defines three concepts `Rocket(x)`, `Truck(x)`, `Cargo(x)`, and two relations, namely `launches(x,y)` that specifies that some `Rocket` launches some `Cargo`, and `moves(x,y)` that describes that some `Truck` moves some `Cargo`. Source S_1 contains predicates in the form `hasPayload(x,y)` related to `rockets` and `cargo` but without explicit class assignment of `x` and `y` variables. Similarly, source S_2 consists of predicates `transports(x,y)` that implicitly link `trucks` and `cargo`.

2.4.1 Local-as-View (LAV) approach

Local-as-View (LAV) [46]

Definition 2.4.2 *In a data integration system $\mathcal{IS} = \langle O, S, M \rangle$ based on LAV approach the mapping M*

associates to each source in S a query q_O in terms of the global schema O:

$$s \rightarrow q_O$$

That is, the sources are represented as a view over the global schema.

LAV is well-suited for the setups where a global schema is stable but sources are frequently updated. Therefore, LAV promotes extensibility as adding a new source only requires a new mapping for this particular source using permanent concepts from the global schema. Hence, in the LAV approach the predicates in S_1 and S_2 are mapped as a conjunctive query over the concepts in the global schema:

```
hasPayload(x,y) := launches(x,y),Rocket(x),Cargo(y)
transports(x,y) := moves(x,y),Truck(x),Cargo(y)
```

`Rocket` and `Cargo` entities provide type assertions for the variables of `hasPayload(x,y)`. Similarly, `Rocket` and `Cargo` explicitly assert types for the variables of `transports(x,y)`.

2.4.2 Global-as-View (GAV) approach

Global-as-View (LAV) [46]

Definition 2.4.3 *In a data integration system $\mathcal{IS} = \langle O, S, M \rangle$ based on GAV approach the mapping M associates to element in the global schema O a query q_S in terms of the sources S:*

$$o \rightarrow q_S$$

That is, the elements of the global schema are represented as a view over the sources.

In contrast to LAV, GAV performs best when a set of sources is stable. In query processing, GAV directly maps concepts from the global schema to local predicates so that simple query unfolding strategy leads to a correct source. However, adding a new source is difficult due to the need to redefine all mappings taking into account new available schema Applying the GAV approach to the global schema O in Fig. 2.5, the concepts are determined as a query over local predicates from S_1 or S_2:

```
launches(x,y) := hasPayload(x,y)
    Rocket(x) := hasPayload(x,y)
     Cargo(y) := hasPayload(x,y)
   moves(x,y) := transports(x,y)
     Truck(x) := transports(x,y)
     Cargo(y) := transports(x,y)
```

Class assertions `Rocket`, `Cargo`, `Truck` are mapped to the arguments `x` and `y` of the predicates `hasPayload(x,y)` and `transports(x,y)`. Note that in GAV several sources might describe one global concept, e.g., `Cargo` instance might be deduced from the both available local predicates.

2.5 Background Overview

The research problem of creating actionable knowledge from Big Enterprise Data and particular research questions stated in Chapter 1 require a comprehensive approach from different angles. The concepts presented in this Chapter lay a solid foundation for addressing the posed challenges.

Fig. 2.6 illustrates which background concepts are relevant for the given tasks. Data Integration System (IS) and integration approaches presented in Section 2.4 define basic principles for solving a task of heterogeneous enterprise data integration. Particularly, we rely on the Local-as-View (LAV) integration paradigm when processing a variety of datasources for a subsequent knowledge creation. In the presence of a stable global schema, i.e., an envisioned RDF knowledge graph, LAV provides a set of advantages in mapping and query processing. Furthermore, LAV enables unified integration of any number of sources in the *plug-and-play* way, i.e., dynamic schema adjustment to newly attached or removed datasources. IS contributes to answering research questions **RQ1** and **RQ2**.

The Resource Description Framework (RDF) described in Section 2.1 serves as a *lingua franca* for knowledge representation obtained from different datasets. As a machine-understandable language with explicit semantics, RDF employs a semi-structured graph-based data model that is more versatile compared to rigid relational models. That is, RDF features fit best for creating and maintaining knowledge graphs. Additionally, expressivity of RDF can be enriched with OWL ontologies that allow intelligent machine agents to perform automated reasoning and deduce new facts out of implicitly encoded semantics. We leverage these characteristics of RDF in research questions **RQ1** and **RQ2**.

The concept of a Knowledge Graph (KG) explained in Section 2.2 is a novel knowledge organization and fusion framework that we employ as a basis for Linked Enterprise Data integration. KG logically connects RDF datasets in a unified view that looks like one whole entity for external applications. Interweaving RDF with KG we devise a concept of an Enterprise Knowledge Graph (EKG) that we leverage in research questions **RQ2** and **RQ3**.

The principles of SPARQL query processing discussed in Section 2.3 bridge a gap between a knowledge graph and real-world applications. Expressivity and query capabilities of SPARQL engines define flexibility and quality of communication interfaces for both humans and machine agents. Moreover, an efficient and effective query engine has to take into account performance trade-offs as well as a configuration of queryable sources. We address research question **RQ3** by applying SPARQL query processing on top of Enterprise Knowledge Graphs.

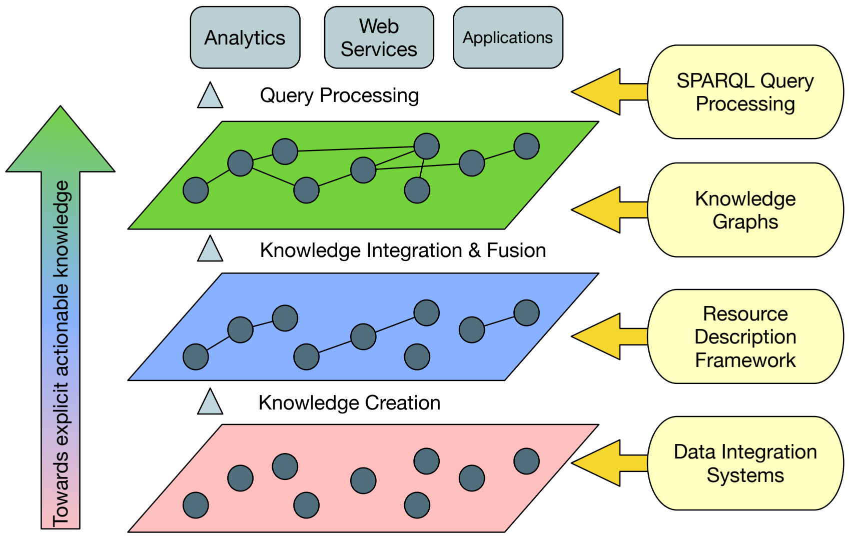


Figure 2.6: **Relevant background concepts for the defined research problem.** Data Integration Systems lay foundations of heterogeneous enterprise data integration. The Resource Description Framework provides a machine-understandable knowledge representation. Knowledge Graphs supply a framework for uniform knowledge integration and fusion. SPARQL Query Processing is responsible for efficient applications of knowledge graphs.

CHAPTER 3

Related Work

This chapter reviews community efforts and state-of-the-art approaches related to the main research problem and research questions. We show that the problem remains largely unexplored while existing work only partly addresses the stated challenges. In Section 3.1 we analyze methodologies and techniques for knowledge acquisition, organization, and management including, e.g., knowledge graphs. The results suggest that knowledge graphs tailored for enterprise needs have not yet gained a necessary momentum that motivates our research in Enterprise Knowledge Graphs (EKG). Section 3.2 describes advances in integrating and fusing RDF graphs hence applicable for EKGs. We observe that whereas knowledge fusion is studied to a very limited extent, RDF graphs integration approaches often do not consider semantics encoded in the graphs and thus still have a potential to be improved. Section 3.3 studies applicability of existing SPARQL query processing techniques in querying RDF knowledge graphs including query decomposition, planning and execution. Due to the Big Data origins of knowledge graphs query engines have to support a set of features, e.g., scalability, federated querying, querying sources with heterogeneous schemas. We take a closer look at query planners and join operators the planners employ in obtaining query answers. We conclude that the potential of knowledge graphs in enterprise settings is yet to be elicited.

3.1 Knowledge Graph Creation Approaches

Knowledge graphs are modeled using various techniques, i.e., not necessarily semantic technologies. Non-semantic approaches stem from the conceptual modeling domain and include, e.g., variations of UML[1] [53, 54], IDEF[2] family of models [55]. However, such approaches can not be utilized considering the stated research problem as they do not provide a formal machine-readable framework thus being useful for humans only. Expert systems relied on rule-based systems and frames proposed by Minsky [56]. As precursors of RDF rules encoded knowledge in *if-then* or *Horn* [57] clauses. CLIPS [58] and Drools [59, 60] are examples of expert systems with reasoning functionality.

Knowledge graphs that adhere to the semantic paradigm, i.e., RDF, are based on public RDF datasources. Those community driven knowledge graphs are either general-purpose bases (e.g., DBpedia [11], Wikidata [12], LinkedGeoData [61], Freebase) which comprise facts from different domains or domain specific bases which aim at providing detailed insights on a particular theme. DBpedia is one of the largest graphs containing more than three billion facts. The source of DBpedia facts are Wikipedia infoboxes. Wikidata is envisioned to be a universal source of information for populating Wikipedia

[1] http://www.omg.org/spec/UML/About-UML/
[2] http://www.idef.com/

articles. Freebase has been acquired by Google as a source for its Knowledge Graph and Knowledge Vault [62]. LinkedGeoData is a geospatial knowledge base that contains semantically processed data collection obtained from OpenStreetMap [3].

Enterprises have integrated Semantic Web technologies with their IT infrastructures in various forms. Statoil used Ontology-Based Data Access (OBDA) to integrate their relational databases with ontologies [63]. NXP Semiconductors transformed product information into an RDF product taxonomy [64]. Stolz et al. [65] derived OWL ontologies from product classification systems. In the Enterprise Architecture (EA) field, which is also concerned with the organization and workflows of enterprise data, Osenberg et al. [66] have tailored a framework for transforming an arbitrary EA model into an ontology. ISO 15926[4] is an ISO standard that proposed an ontological model for modeling oil and gas industrial routines. All the above-mentioned efforts have, however, only analyzed the one specific domain that is directly relevant, and although they refer to KGs profusely the authors do not elaborate on the definition, conceptual description or reference architecture of such KGs. In general, a precise definition of the Enterprise Knowledge Graph concept remains to be developed. An incorrect positioning of such technologies with respect to existing enterprise applications also impedes the industrial appraisal of enterprise semantic technologies. An example of this effect is the frequent interchangeable (and incorrect) use of Master Data Management (MDM) systems and KBs (and subsequently KGs), which reflects a lack of awareness by data management executives. Considering MDM systems and data quality in the Business Informatics field, the Competence Center Corporate Data Quality (CC CDQ) has developed a framework [67] for Corporate Data Quality Management (CDQM) that presents a model for the evaluation of data quality within a company. Being a thoroughly tailored enterprise tool, the framework is, however, only applicable to conventional data architectures, i.e., MDM, Product Information Management (PIM) and Product Classification Systems (PCS), and extending the framework to cover semantic data architectures is impractical. Maturity models or capability models are widely-used means for assessing an abstract level of applicability for certain technologies or methodologies. There exist maturity models in the software engineering domain, namely Capability Maturity Model Integration (CMMI)[5]; in the data governance domain[6]; in the enterprise data quality domain [68]. Maturity models for EKGs have not yet been developed.

3.1.1 Knowledge Acquisition from Web Tables

Enterprise data on the Web is often shipped in spreadsheets and various tabular formats, e.g., simple HTML tables that are not well structured and lack semantics. The problem of extracting knowledge from HTML tables is discussed in the paper [69] that concerns methods of acquiring datasets related to roads repair. There is also a raw approach [70] in information extraction, which is template-based and effective in processing of web sites with unstable markup. The crawler was used to create a LOD dataset of CEUR Workshop[7] proceedings.

Silva et al. in their paper [71] suggest and analyze an algorithm of table research that consists of five steps: location, segmentation, functional analysis, structural analysis and interpretation of the table. The authors provide a comprehensive overview of the existing approaches and designed a method for extracting data from ASCII tables. However, smart tables detection and distinguishing is not considered.

[3]https://www.openstreetmap.org/

[4]http://15926.org/

[5]CMMI for development: http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=9661

[6]The Data Governance Maturity Model. Whitepaper: http://www.fstech.co.uk/fst/whitepapers/The_Data_Governance_Maturity_Model.pdf

[7]Web: http://ceur-ws.org/

Hu et al. introduced in the paper [72] the methods for table detection and recognition. Table detection is based on the idea of edit-distance while table recognition uses random graphs approach. M. Hurst takes into consideration ASCII tables [73] and suggests an approach to derive an abstract geometric model of a table from a physical representation. A graph of constraints between cells was implemented in order to determine position of cells. Nevertheless, the results are rather high which indicates the efficiency of the approach. The authors of the papers achieved significant success in structuring a table, but the question of the table content and its semantic is still opened.

Embley et al. tried [74] to solve the table processing problem as an extraction problem with an introduction of machine learning algorithms. However, the test sample was rather small which might have been resulted in overfitting [75].

Gatterbauer et al. in the paper [76] developed a new approach towards information extraction. The emphasis is made on a visual representation of the web table as rendered by a web browser. Thus the problem becomes a question of analysis of the visual features, such as 2-D topology and typography.

Tijerino et al. introduced in [77] the TANGO approach (Table Analysis for Generating Ontologies) which is mostly based on WordNet [78] with a special procedure for ontology generation. The whole algorithm implies 4 actions: table recognition, mini-ontology generation, inter-ontology mappings discovery, merging of mini-ontologies. During the table recognition step the search in WordNet supports the process of table segmentation so that no machine learning algorithms were applied.

Crescenzi et al. introduced in [79] a *RoadRunner* algorithm for an automatic extraction of the HTML tables data. The process is based on regular expressions and matches/mismatches handling. Eventually a DOM-like tree is built which is more convenient for an information extraction. A comprehensive Karma [80] framework for semantic processing of HTML pages emerged from this work.

To sum up, there are different approaches to information extraction developed last ten years. In our work we introduce an effective extraction and analyzing framework built on top of those methodologies combining tables recognition techniques, machine learning algorithms and Semantic Web methods.

Automatic data extraction has always been given a lot of attention from the Web community. There are numerous web-services that provide users with sophisticated instruments useful in web scraping, web crawling and tables processing. Some of them are presented below.

ScraperWiki[8] is a tool that is suitable for software engineers and data scientists whose work is connected with processing of large amounts of data. Being a platform for interaction between business, journalists and developers, ScraperWiki allows users to solve extracting and cleaning tasks, helps to visualize acquired information and offers tools to manage retrieved data. Some of the features of the service: 1) Dataset subscription makes possible the automatic tracking, update and processing of the specified dataset in the Internet. 2) A wide range of data processing instruments. For instance, information extraction from PDF documents ScraperWiki allows one to parse web tables in CSV format, but processes all the tables on the page even thought they do not contain re levant data, e.g. layout tables. Additionally, the service does not provide any Linked Data functionality.

Scrapy[9] is a fast high-level library written in Python for web-scraping and data extraction. Scrapy is distributed under BSD license and available on Windows, Linux, MacOS and BSD. Combining performance, speed, extensibility and simplicity Scrapy is a popular solution in the industry. A lot of services are based on Scrapy, e.g., the above-mentioned ScraperWiki.

Import.io[10] is an emerging data processing service that provides comprehensive visualizations and an opportunity to use the service without programming experience. The system offers users three methods

[8]Web: `https://scraperwiki.com/`
[9]Web: `http://scrapy.org/`
[10]Web: `https://import.io/`

of extraction arranged by growing complexity: an extractor, a crawler and a connector. The feature of automatic table extraction is also implemented but supports only CSV format.

Concluding the paragraph, we observe a lack of methods and tools for extracting formalized knowledge from Web tables. Tabular data remains to be one of the main formats for representing enterprise data. Therefore, novel approaches for knowledge acquisition have to be developed. We present our methodology in Section **??** in Chapter 4.

3.2 RDF Graph Integration Efforts

Having discussed knowledge extraction in Section 3.1.1, in this section, we study how the community tackled the problem of heterogeneous RDF graph integration and fusion which are two other major components of semantic data integration systems. We also cover a range of similarity measures used in semantic data integration.

3.2.1 End-to-End Integration

By the *end-to-end* integration, we understand systems that receive several datasources as an input and produce an integrated RDF dataset as an output. Large volumes of common datasets (e.g., DBpedia dump is about 250 GB, PubMed [11] dump is about 1.6 TB) imply, that the integration pipeline has to be as automatic as possible. Moreover, traditional approaches towards constructing KGs, e.g., NOUS [81], DeepDive [82], NELL [83], or Knowledge Vault [84], imply materialization of the designed graph built from (un-,semi-)structured sources which results in high memory and space consumption. Tackling time dimension requires even more resources [85]. Prominent approaches for automatic linked data integration and fusion are Linked Data Integration Framework (LDIF) [86], OD CleanStore [87], LIMES [88], Karma [80]. All of them maintain an integration pipeline starting from data ingestion from various remote data sources to a high-quality target data source through identification of similar entities [89]. LDIF is the framework that provides a set of independent tools to support the process of interlinking RDF datasets. For instance, SILK [90] identifies `owl:sameAs` links among entities of two datasets and Sieve [91] performs data fusion resolving property values conflicts by the assessment of the quality of the source data and by application of various fusion policies. OD CleanStore [92] is an RDF data integration approach which relies on Silk when performing instance matching and involves a custom data fusion module when merging discovered matches. LIMES [88] is a stand-alone link discovery tool which utilizes mathematical features of metric spaces in order to compute similarities between instances. Sophisticated algorithms significantly reduce the number of necessary comparisons of property values. UnifiedViews [93] is an Extract-Transform-Load (ETL) framework for processing RDF data. Based on ODCleanStore, UnifiedViews supports a wide range of processing tasks including instance linking and data fusion (the LD-FusionTool [94] is responsible for data fusion).

Although the aforementioned approaches are fast and effective, they require domain knowledge and significant manual effort while configuring the pipeline. Furthermore, those approaches belong to the *triple-based* integration paradigm where the main unit of integration is an RDF triple. The focus on triples impedes understanding of a bigger picture, i.e., the whole RDF entity comprised of multiple RDF triples that encode rich semantics about the given entity. As previous work has focused mostly on the problem of heterogeneity of RDF schemas in different data sources, in integrating RDF graphs the

[11] The database of references and abstracts in the life sciences and biomedical domain. https://www.ncbi.nlm.nih.gov/pubmed/

problem of identifying relevant entities has to be considered. That is, the inherent semantics encoded in an RDF graph has to be leveraged in order to increase the quality of the integration.

3.2.2 Entity Interlinking in Similarity Operators

Whereas end-to-end approaches aim at creating a knowledge graph performing all necessary actions in their own *black box*, modular pipeline architectures allow to use various components for different tasks. That is, similarity operators and similarity functions cater for the entity interlinking task. Entity interlinking aims at identifying similar or equivalent entities and establish a link between them in order to reduce data duplication and promote data fusion. Similarity functions produce a numerical value of relatedness between two entities. Similarity operators are often applied directly during query processing and employ various similarity functions. Existing federated SPARQL query engines implement Boolean join operators to merge RDF graphs from different data sources as an answer of a SPARQL query. One of such approaches is ANAPSID [95] which introduces two Boolean operators, the adaptive group join (*agjoin*) and adaptive dependent join (*adjoin*). FedX [96] describes the *bound join* technique which uses one subquery to evaluate the input sequences. SPLENDID [97] provides *hash joins* and *bind joins* to optimize the performance in the query execution strategies when the join arguments are processed in parallel from the data sources (*hash join*) or a variable must be bound for the succeeding join operation (*bind join*). Ladwig et al. [98] propose an operator called *SIHJoin*, a symmetric index hash join that addresses the problem of Linked Data query processing not only remotely but also locally. However, traditional binary join operators require join variables instantiations to be exactly the same. For example, XJoin [99] and Hash Join [45] operators abide this condition. At the *Insert* step, both blocking and non-blocking Hash Join algorithms partition incoming tuples into a number of buckets based on the assumption that after applying a hash function similar tuples will reside in the same bucket. The assumption holds true in cases of simple data structures, e.g., numbers or strings. However, applying hash functions to string representations of complex data structures such as full descriptions of RDF entities tend to produce more collisions rather then efficient partitions. At the *Probe* stage, Hash Join performs matching as to a specified join variable. Thus, having URI as a join variable, semantically equivalent entities with different URIs can not be joined by Hash Join.

Similarity join algorithms are able to match syntactically different entities and address the heterogeneity issue. String similarity join techniques reported in [100–102] rely on various metrics to compute a distance between two strings. Set similarity joins [103, 104] identify matches between sets. String and set similarity techniques are, however, inefficient being applied to RDF data as they do not consider the graph nature of semantic data. There exist graph similarity joins [105, 106] which traverse graph data in order to identify similar nodes. On the other hand, those operators do not tackle semantics encoded in the knowledge graphs and are tailored for specific similarity functions.

Therefore, we foresee the need for an algorithm that would fully leverage RDF and OWL semantics encoded in RDF graphs. Moreover, such an algorithm has to have implementations that perform in blocking, i.e., 1-1 perfect matching, conditions or non-blocking, i.e., incremental $N - M$, manner allowing for on-demand and ad-hoc semantic data integration pipelines. It is also advisable to support various similarity functions and metrics, e.g., from simple *Jaccard* similarity to complex *NED* [107] or *GADES* [108] functions that drastically increase versatility and applicability of the algorithm.

3.3 SPARQL Federated Query Processing Approaches

3.3.1 Federated Query Engines

Saleem et al. [109] study federated query engines with native Web access interfaces. *ANAPSID* [95] is an adaptive federated query engine capable to adjust the execution depending on the detected traffic load of a SPARQL endpoint. The engine delivers query results as soon as they arrive from sources, i.e., by implementing non-blocking join operators. *FedX* [96] is a non-adaptive federated query processing approach. FedX optimizes query execution introducing exclusive groups, i.e., triple patterns of a SPARQL query which have only one relevant source. In contrast to ANAPSID, FedX does not require metadata about the sources beforehand, but uses ASK queries. *Avalanche* [110] is a federated query engine that requires no prior knowledge about sources (e.g. SPARQL endpoints), their content, data schema, or accessibility of the endpoints. Instead, Avalanche identifies relevant sources and plans the query based on online statistical information. *Linked Data Fragments* (LDF) [38] is an approach that provides distributed storage and federated querying element optimized for processing *triple patterns*.

Federated query engines rely on source descriptions for source selection. Görlitz et al. describe SPLENDID, a federated query engine which leverages endpoint metadata available as *Vocabulary of Interlinked Datasets* (VoID) [111]. Although VoID provides dataset statistics, its description is limited and lacks details necessary for efficient query optimization. *HiBISCuS* [112] is a source selection technique based on hypergraphs. HiBISCuS discards irrelevant sources for a particular query by modeling SPARQL queries as hypergraphs.Capability metadata is gathered for each endpoint to label and subsequently prune a query hypergraph. FEDRA [113] is a source selection strategy for sources with a high replication degree. Nevertheless, leveraging multiple Web access interfaces in one query answering pipeline was largely unexplored. Wylot and Cudré-Mauroux [114] use a notion of RDF subgraphs but provide only an intuitive description without formalization.

As to query decomposition, *FED-DSATUR* presented in [115] applies graph colouring algorithms to perform query decomposition. *LILAC* [116] is a query decomposition technique that tackles data replication across data sources and SPARQL endpoints.

We identify a significant technological granularity in source selection, query decomposition, and partitioning stages. Leveraging semantics for enhancing federated query processing still poses numerous challenges. Therefore, we see here a room for algorithmic improvements that are crucial for scalable query processing as stated in the **RQ3**.

3.3.2 Join Operators

Binary join operators are used by both Database and Semantic Web query processing engines. The operators are often classified along several dimensions, e.g., blocking and non-blocking, adaptive and non-adaptive. Federated query engines often employ a collection of join operators at hand to choose the best relevant and most efficient operator for a current condition.

Nested Loop Join [117], Hash Join [45], and Symmetric Hash Join (SHJ) [45] are traditional binary join operators that rely on hash tables as inner data structures to reduce the amount of necessary comparisons between tuples. Additionally, SHJ implements a non-blocking pipeline. XJoin [118] and Dependent join [119] are adaptive binary join operators that are able to adjust its behavior to address a case when a source becomes blocked or delayed.

SPARQL federated query processing engines extend traditional join operators. Firstly, ANAPSID [95] introduces the Adaptive Group Join (*agjoin*) algorithm on top of XJoin and Adaptive Dependent Join (*adjoin*) on top of Dependent join. Secondly, FedX [96] describes the Controlled Bound Worker Join

(CBJ) operator. Symmetric Indexed Hash Join (*SIHJoin*) [120] has been proposed to facilitate querying of both remote and local Linked Data sources. nLDE [121] is a novel approach for adaptive query processing. nLDE dynamically re-arranges a query plan in presence of data transfer delays and routes intermediate results to reduce execution on time. However, all the described engines construct query plans of binary join operators, e.g., Nested Loop Joins or *agjoin*.

Multi-way join operators are well-known in the relational database community [122, 123]. *i*) Common join operators, e.g., *MJoin* [122] based on binary *XJoin* [45] is a starting point for later approaches; *ii*) Sequential joins that adhere to the theoretical boundaries described by Atserias, Grohe and Marx (AGM [124]), e.g., Leapfrog Triejoin [125], a 3-way join operator; *iii*) Parallel joins that often rely on MapReduce to optimize the algorithm performance, e.g., *GrubJoin* [126], *Theta-Join* [127]. However, such operators are not suited nor applicable for SPARQL.

SigMR [128] is a query processing system that tries to combine multi-way joins with SPARQL and MapReduce paradigm. Albeit proposing a multi-way join operator, it is bound to MapReduce and therefore blocking, and has to compute matrices of unnecessary comparisons and prune the results set. CQELS [129] implements a windowed multi-way join operator for RDF streams processing.

To the best of our knowledge, none of existing SPARQL engines provides a planner capable of employing both binary and multi-way join operators in one query plan. Centralized query engines of state-of-the-art triple stores, e.g., *Virtuoso* [130], *Apache Jena Fuseki*[12], RDF4J[13], GraphDB (former OWLIM) [131] consider only binary join operators in query processing. Nor multi-way joins are supported by federated and distributed query engines, e.g., ANAPSID [95], FedX [96], Avalanche [110], Triple Pattern Fragments Client [38], Strider [132].

Having analyzed state-of-the-art in SPARQL query engines and join operators we deduce that multi-way join operators that are already embraced by the relational database community for their performance in a certain subset of queries remain strangers in SPARQL. In contrast to relational engines, there is no adaptive query engine in the semantic data management domain that is able to utilize multi-way joins together with binary joins in order to produce effective SPARQL query plans. This fact impedes scalability of SPARQL query engines as to the question stated in the **RQ3** of this thesis.

[12]https://jena.apache.org/documentation/fuseki2/index.html
[13]http://docs.rdf4j.org/

CHAPTER 4

Semantic Representation and Integration of Enterprise Data in a Knowledge Graph

In the previous chapters, we posed research problems, challenges, and questions. We also illustrated that there is a clear lack of semantic frameworks for representing enterprise knowledge. This Chapter is dedicated to Level 1 and Level 2 of the problem space (cf. Fig. 4.1), i.e., creating, representing, and integrating knowledge obtained from raw data about real world concepts. We study how non-machine-understandable data full of implicit links and inherent semantics can be explicitly represented as machine-understandable knowledge. The following research question is addressed in this Chapter:

Research Question 1 (RQ1)

How can we apply the concept of knowledge graphs for representing enterprise information to become actionable knowledge?

We present the concept of an *Enterprise Knowledge Graph (EKG)*, a semantic framework suited for enterprise knowledge management. The EKG facilitates first steps towards explicit actionable knowledge. On the lower level, the EKG adopts the Local-as-View (LAV) data integration approach serving as a global schema for heterogeneous data sources. On the higher level, the EKG employs the graph-based RDF model in order to provide a unified, semantic, machine-understandable language for creating the global schema as well as representing knowledge extracted from the data sources. Backed up by

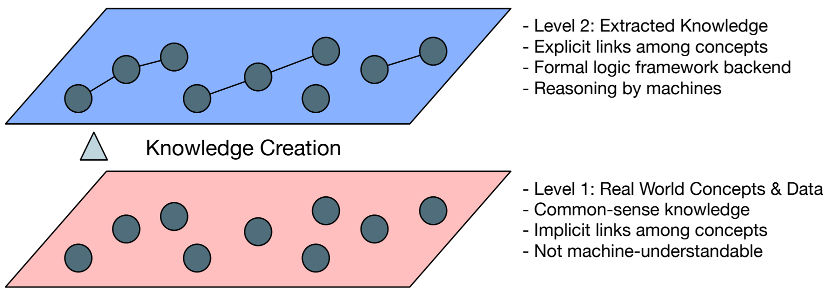

- Level 2: Extracted Knowledge
- Explicit links among concepts
- Formal logic framework backend
- Reasoning by machines
Knowledge Creation
- Level 1: Real World Concepts & Data
- Common-sense knowledge
- Implicit links among concepts
- Not machine-understandable

Figure 4.1: **The levels of the research problem addressed in this Chapter.** We tackle challenges concerning knowledge creation, its semantic representation, and integration.

logic mechanisms and extensible ontologies, the EKG enables automated reasoning that is leveraged at query processing and analytic stages. That is, the EKG presents a *black box* view on the knowledge for higher-level services separating integration pipelines from applications. To illustrate knowledge acquisition for an EKG, we devise a semantic approach for Web tables processing as spreadsheets and tabular formats remain to be one of the main data representation layouts used by enterprises.

The contributions of this chapter towards the stated **RQ1** and the research problem in general:

- Formal definition and description of the *Enterprise Knowledge Graph* concept, study how they are able to grasp and represent knowledge in the graph-based RDF structure;
- A formal framework for comparing EKG-based approaches aimed at deriving main principles when developing EKGs;
- Analysis of existing enterprise knowledge management systems and their comparison against an EKG;
- Analysis of the impact of EKG features on query performance to demonstrate feasibility of large-scale processing which is studied later in Chapter 6;

Chapter 4 is based on the following publications [133–135].

The Chapter is structured is as follows. Section 4.1 presents the EKG concept. In Section 4.2 we compare EKG with existing solutions that partly implement some of EKG functionality and discuss experiments conducted on an EKG prototype. Finally, in Section 4.3 we examine the research question and provide conclusion remarks with an outlook to future work.

4.1 Enterprise Knowledge Graphs

In enterprises, Semantic Web technologies have recently received increasing attention from both the research and industrial side. The concept of Linked Enterprise Data (LED) describes a framework to incorporate benefits of Semantic Web technologies into enterprise IT environments. However, LED still remains an abstract idea lacking a point of origin, i.e., station zero from which it comes to existence. We devise Enterprise Knowledge Graphs (EKGs) as a formal model to represent and manage corporate information at a semantic level. EKGs are presented and formally defined, as well as positioned in Enterprise Information Systems (EISs) architectures.

The demand for new Knowledge Management (KM) technologies in enterprises is growing in recent years [136]. The importance of KM increases with the volumes of data processed by a company. Although the enterprise domain might vary from car manufacturing to software engineering, KM is capable of reducing costs, increasing the performance and supporting an additional added value to company's products. Novel KM approaches often suggest new data organization architectures and foster their implementation in enterprises. One of such an architecture leverages semantic technologies, i.e., the technologies the Semantic Web is based on, in order to allow machines to understand the meaning of the data they work with. Machine understanding and machine-readability are supported by complex formalisms such as description logics and ontologies. Semantic applications in the business domain comprise a new research trend, namely Linked Enterprise Data (LED). However, in order to truly exploit LED an organization needs to establish a knowledge hub as well as a crystallization and linking point [137]. Google, for example, acquired Freebase and evolved it into its own Enterprise Knowledge Base [138], whereas DBpedia assumed a similar position for the Web of Linked Data overall [139].

Table 4.1: Comparison of various Enterprise Data Integration Paradigms: P1:XML Schema Integration, P2:Data Warehouses, P3:Data Lakes, P4:MDM, P5: PIM/PCS, P6:Enterprise Search, P7:EKG. Checkmark and cross denote existence and absence of a particular feature, respectively

Para-digm	Data Model	Integr. Strategy	Conceptual/ operational	Heterogen-eous data	Internal/ ext. data	No. of sources	Type of integr.	Domain coverage	Semantic repres.
P1	DOM trees	LAV	operational	✓	✓	medium	both	medium	high
P2	relational	GAV	operational	✘	partially	medium	physical	small	medium
P3	various	LAV	operational	✓	✓	large	physical	high	medium
P4	UML	GAV	conceptual	✘	✘	small	physical	small	medium
P5	trees	GAV	operational	partially	partially	✘	physical	medium	medium
P6	document	✘	operational	✓	partially	large	virtual	high	low
P7	RDF	LAV	both	✓	✓	medium	both	high	very high

In this section, we present Enterprise Knowledge Graphs (EKGs), as formal models for the embodiment of LED. By EKG we refer to a semantic network of concepts, properties, individuals and links representing and referencing foundational and domain knowledge relevant for an enterprise. EKGs offer a new data integration paradigm that combines large-scale data processing with robust semantic technologies making first steps towards the next generation of Enterprise Information Systems [140]. We define an EKG concept in light of data integration systems and RDF knowledge graphs, and provide an extensive study of existing Enterprise Information Systems (EISs), which offer certain EKG functions or can be used as a basis for an EKG. To achieve such a goal, we develop an independent assessment framework which is presented in the paper as well. We report on an unsupervised evaluation that allows for clustering existing EISs in terms of EKG features. Observed results give evidences that none of state-of-the-art approaches fully supports EKGs, and further study is required in order to catch the wave of future EISs. To achieve such a goal, we developed a universal assessment framework that takes into account the most important features in three crucial dimensions: a human interaction dimension consists of means employed by humans, e.g., architects or users, to design, visualize and explore the contents of an EKG; a machine interaction dimension specifies features, which foster automatic communication among enterprise applications via an EKG; and a strategic development dimension, which regulates and standardizes the EKG interaction pipeline in large multi-user enterprise environments.

4.1.1 Motivation

In the last decades a variety of different approaches for enterprise data integration have been developed and deployed. Table 4.1 shows a comparison of the main representatives according to the following criteria: 1) *Data Model*: Various data models are used by data integration paradigms. 2) *Integration Strategy*: The two prevalent integration strategies are *Global-As-View* (GAV), where local sources are viewed in the light of a global schema and *Local-As-View* (LAV), where original sources are mapped to the mediated/global schema. 3) *Conceptual versus operational*: Some approaches primarily target the conceptual or modeling level, while the majority of the approaches are aiming at supporting operational integration, e.g., by data transformation or query translation. 4) *Heterogeneous data*: Describes the extent to which heterogeneous data in terms of data model or structure is provided. 5) *Internal/external data*: Describes whether the integration of external data is supported in addition to internal data. 6) *Number of sources*: Presents the typically provided number of sources, i.e., small (less than 10), medium (less than 100), large (more than 100). 7) *Type of integration*: States whether data is physically integrated and materialized in the integrated form or virtually integrated by executing queries over the original data. 8) *Domain coverage*: Reports the typical coverage of different domains. 9) *Semantic representation*: Expresses the extent of semantic representations, which can be low in the case of unstructured or semi-

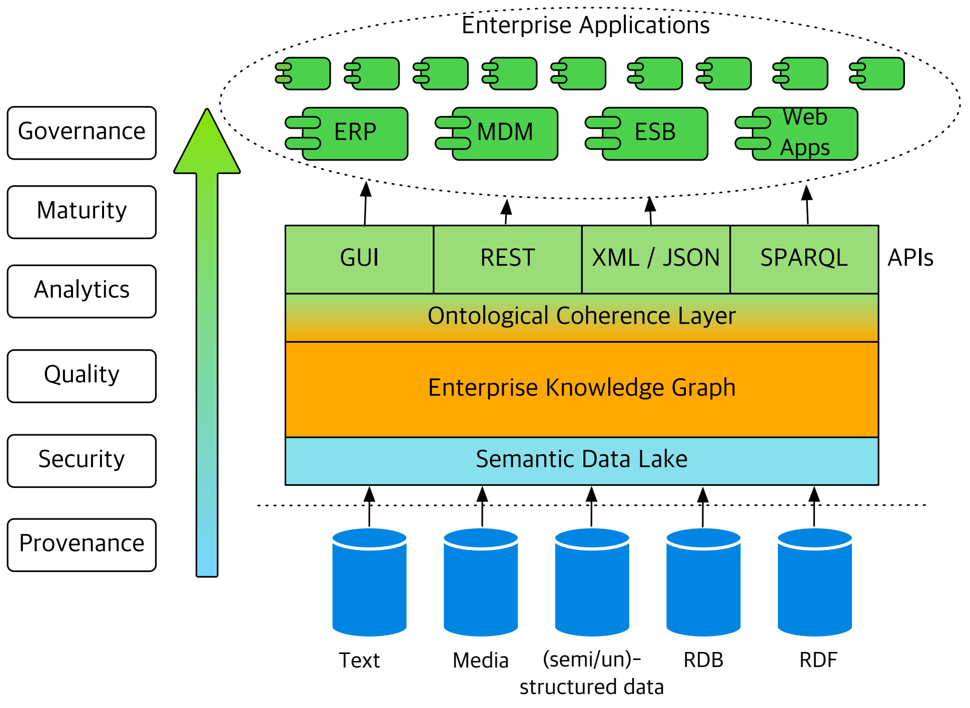


Figure 4.2: **Position of an EKG in the Enterprise Information System architecture.** An EKG assumes a mediate position between raw enterprise data storage and numerous services whereas the coherence layer provides a unified view on the data

structured documents, medium in the case of relational or taxonomic data and high, when comprehensive semantic formalisms and logical axioms are supported.

The comparison according to these criteria shows, that the EKG paradigm differs from previous integration paradigms. EKGs are based on the statement-centric RDF data model, which can mediate between different other data models, since, for example, relational, taxonomic or tree data can be easily represented in RDF. Similar to XML-based integration and the recently emerging Data Lake approach, EKGs follow the Local-As-View integration strategy and are thus more suited for the integration of a larger number of possibly evolving sources. EKGs are the only approach, which bridges between conceptual and operational data integration, because on the one hand ontologies and vocabularies are used to model domain knowledge, but at the same time operational aspects, such as querying and mapping/transformation are supported. By employing RDF with URI/IRI identifiers, EKGs support the integration of heterogeneous data from internal and external sources either in a virtual or physical/materialized way. EKGs support the whole spectrum of semantic representations from simple taxonomies to complex ontologies and logical axioms. However, a promising strategy seems to be to use EKGs in combination with other approaches. For example, EKGs can be used in conjunction with a data lake, to add rich semantics to the low level source data representations stored in the data lake. Similarly, EKGs can provide background knowledge for enriching enterprise search or EKGs can comprise vocabularies and mappings from MDM.

Thus, EKGs occupy a unique niche in the ecosystem of enterprise applications. In order to provide a better understanding of such a niche, we present a structural view that represents a static orchestration of information systems in a company (cf. Figure 4.2).

The structural view shows an EKG as a consumer of raw heterogeneous data and a supplier of knowledge for enterprise applications. Numerous structured, e.g., relational data in various RDB formats, annotated data or semantic data in RDF, and unstructured data sources, e.g., texts in natural language, media files, documents, compose a new form of a data lake, i.e., a Semantic Data Lake (SDL). In addition to common data lake characteristics, e.g., high scalability, an SDL features semantic metamodels and interfaces allowing for the on-the-fly data integration. An SDL can integrate both company's private data as well as remote data in the open access, e.g., Linking Open Data Cloud. An EKG might rely on SDL capabilities for source selection, query composition, query optimization and query execution. The Ontological Coherence layer contains a set of ontologies, both high-level and domain specific, which offer different semantic views on the EKG contents. For example, an organizational ontology defines a business department in a company, a security ontology places access restrictions on the data available to this department, whereas some supply chains ontology specifies the role of the department in a product lifecycle. Such a granularity increases knowledge representation flexibility and ensures that external applications use a standardized, ontology-based view on the required data from the EKG. The API layer exposes communication interfaces, e.g., graphical user interfaces (GUI), REST, XML, JSON, or SPARQL, to deliver knowledge encoded in EKGs to enterprise applications. Possible beneficiaries of using EKG technology include: Enterprise Resource Planning (ERP) systems, Master Data Management (MDM) systems, Enterprise Service Buses (ESB), E-Commerce and a company's Web applications. An EKG specifies a set of domain-independent features, e.g., provenance, governance, or security, which support the operation and maintenance of the entire system. We argue that these enterprise characteristics distinguish an Enterprise Knowledge Graph from a common RDF knowledge graph; these features are explained in detail in the Section of EKG Technologies. Being 'orthogonal' to the knowledge acquisition and representation pipelines, the features are an integral part of the knowledge management policies. We also elaborate on those enterprise features in the Section of EKG Technologies.

4.1.2 EKG Definition

We define Enterprise Knowledge Graphs (EKGs) as a part of an Enterprise Data Integration System (EDIS). Extending the Definition 2.4.1 we put emphasis on the global schema and sources parts.

Enterprise Data Integration System

Definition 4.1.1 *Formally, EDIS is defined as a tuple:*

$$EDIS = \langle EKG, S, M \rangle = \langle \langle \sigma, A, R \rangle, \langle OS, PS, CS \rangle, M \rangle \tag{4.1}$$

An EKG contains ontologies and instance data, S represents data sources, and M represents the collection of mappings to translate S into an EKG. EKGs are based on the directed graph model $\langle N, E \rangle$ where nodes N are entities and edges E are relations between the nodes. OS, PS, CS are open sources, private sources, and closed sources, respectively.

Considering Definition 2.2.1 and Definition 2.2.2 we define an Enterprise Knowledge Graph as a particular kind of an RDF knowledge graph.

Enterprise Knowledge Graph

Definition 4.1.2 *An EKG is defined as a tuple:*

$$EKG = \langle \sigma, A, R \rangle, \tag{4.2}$$

where σ is a signature *for a logical language, A is a collection of* axioms *describing an ontology, and R is a set of* restrictions *on top of the ontology.*

A signature σ is a set of relational and constant symbols which can be used to express logical formulas. In other words, a signature contains definitions of entities, e.g., the RDF triples `:ToolX a owl:Class` and `:Product a owl:Class`[1].

The axioms in the set A provide additional description of the ontology by defining the relationships between the concepts defined in σ. In more detail, axioms leverage logical capabilities of the chosen ontology development methodology, e.g., RDFS allows for (but is not limited to) class hierarchy as well as domain and range definitions of properties. For instance, `:ToolX rdfs:subClassOf :Product`.

The restrictions R impose constraints on concepts and relationships. Although restrictions are also axioms, i.e., $R \sqsubset A$, we distinguish them as a separate important criteria necessary for large-scale multi-user enterprise environments. Restrictions can be expressed as triples or in a rule language, e.g., *Semantic Web Rule Language* (SWRL) or *SPIN*. Restrictions can be imposed on numerous characteristics: access rights, privacy, or provenance. To illustrate, suppose an RDF triple `:ToolX :cost 100` represents the cost of a tool `ToolX`. Then an RDF triple `:cost :editableBy :FinancialDept` is the restriction which states that only the financial department can change the value of the property `:cost`.

Data sources S are defined as a tuple $\langle OS, PS, CS \rangle$, where OS is a set of open data sources which an enterprise considers appropriate to re-use or publish, e.g., Linked Open Data Cloud or annual financial reports. PS is a set of private data sources of limited access. Supply chain data is an ample example of a private data. CS is a set of closed data sources with the strongest access limitations. Closed data is often available only to a special group of people within a company and hardly ever shared, e.g., technical innovations, business plans, or financial indicators.

Mappings in M connect data sources S with the EKG. Whereas EKG employs an RDF data model, sources may have various different formats. As EKG is a component of EDIS, we use a notion of *conjunctive queries* to give examples of mappings. The sources expose a semantic description of their contents. For instance, let $S_1 = \{producedFrom(x, y)\}$ return tuples that a certain product x is produced from a certain material y. Suppose EKG contains the following RDF triples: `{componentOf a rdf:Property. material a Class. product a Class}` . In order to query the tuples, one should provide mappings between the global ontology in the EKG and the sources. We advocate that the Local-As-View (LAV) paradigm is preferred in the EDIS. According to the LAV paradigm, sources are defined in terms of a global ontology [141]: for each source S_i, there is a mapping that describes S_i as a *conjunctive query* on the concepts in the global ontology EKG that also distinguish input and output attributes of the source. Therefore, M will contain a rule: $producedFrom(x, y) : -componentOf(y, x), material(y), product(x)$. Although there exists a Global-As-View (GAV) paradigm which implies presentation of the global ontology concepts in terms of the sources, LAV approach is designed to be used with numerous changing sources and a stable global ontology. This is exactly the

[1]We use "`a`" as shortcut for `rdf:type` as in the RDF Turtle notation for readability.

EDIS case where an EKG remains stable but enterprise data is highly dynamic; thus, we underline the efficiency of LAV mappings for EKGs.

4.1.3 Creating EKGs

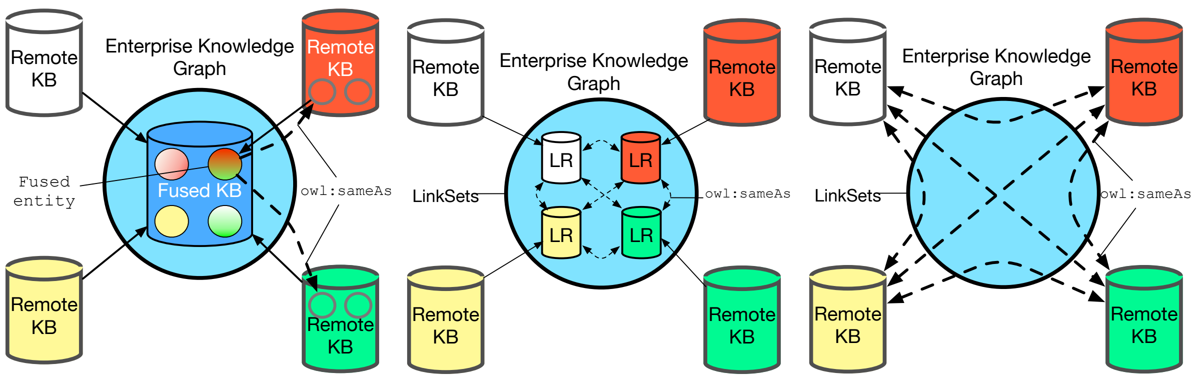
In this paragraph, we aim to join the dots between the expanding interest in EKGs expressed by those communities and the lack of blueprints for realizing the EKGs. A thorough study of the key design concepts provides a multi-dimensional aspects matrix from which an enterprise is able to choose specific features of the highest priority. We emphasize the importance of various data fusion approaches, e.g., centralized *physical integration* and federated *virtual integration*.

Large enterprises and organizations use a vast variety of different information systems, databases, portals, wikis and knowledge bases [KBs] combining hundreds and thousands of data and information sources. Despite a long tradition of efforts addressing the data integration challenge comprising enterprise architecture, master data management, data warehouses, mediators and service oriented architectures, the integration of such enterprise information and datasources is still a major challenge [142]. The publishing and consumption of vocabulary and URI-based Linked Data adhering to the RDF data model is meanwhile seen as a promising strategy to increase coherence and facilitate information flows between various enterprise information systems [143]. Employing the Linked Data paradigm for enterprise data integration has a number of advantages: *Identification* using URI/IRIs enterprise concepts, data and metadata can be uniquely identified across departments, branches, subsidiaries. *Access* – dereferencable URI/IRIs provide a unified data access mechanism. *Integration* – the triple based RDF data model facilitates the integration and mapping between various other data models (e.g., XML, RDB, JSON). *Coherence* – schema, data and metadata can be seamlessly interlinked across system and organizational boundaries. *Provenance* – is ensured, since the origin of information is encoded in the identifiers. *Governance* – identifiers, metadata and schema can be governed in an incremental and decentralized manner. *Agility* – vocabularies, vocabulary elements and datasources can be added incrementally in a pay-as-you-go fashion.

From an abstract point of view, an EKG consists of the following layers: *Schema layer* – comprises vocabularies and ontologies which provide a formal machine readable description of accumulated knowledge; *Instance layer* – comprises description of relevant domain specific and common sense instance data relevant for the enterprise; *Metadata layer* – comprises information about other information sources, knowledge and data bases thus facilitating data discovery; *Coherence layer* – comprises links to other knowledge sources and other knowledge graphs within the enterprise and on the Data Web;

Data and knowledge integration approaches

One of the key problems of the EKGs design is how to accumulate and represent knowledge. Although there are various methods of organizing an EKG, we distinguish among three divergent approaches, the *physical* and the *virtual* approach, each of which represents the two extremes in the spectrum of possibilities, and the *transitionary* approach which balances in between those extremes. A physically integrated EKG creates a new and tightly-integrated fused knowledge graph, which initially bootstraps the EKG with information from other sources, but later evolves independently. On the opposite side of the spectrum, a virtually integrated EKG represents a loosely coupled set of remote datasources and vocabularies, which are replicated and kept in sync with their original representations. By thoroughly discussing options and design choices for answering the above questions along these two strategies, we also outline possible middle courses balancing differently between the advantages and shortcomings of the two, namely the transitionary EKG architecture. The high-level conceptual distinction among



(a) Physical integration (b) Semantically federated approach (c) Virtual integration

Figure 4.3: **Schematic depiction of the EKG sources integration approaches.** Physical integration implies transformation of sources into one unified local storage, e.g., more of a data warehouse approach. Semantically federated, or transitionary, approach implies retaining replicas of remote sources within local infrastructure but keeping their original formats relying on schema mappings. Virtual integration implies all remote sources are treated in their original states and formats with the help of schema mappings.

the approaches is depicted in Fig. 4.3. The left-hand side illustrates how a fused EKG generates new concepts or entities to link the original entities in distributed sources. Links between the new and original entities are retained by use of equivalence statements, such as the *owl:sameAs* property. The right-hand side shows how a federated EKG only consists of a LinkSet that directly links entities which are known to be equivalent. The middle part, i.e., a transitionary approach, combines original datasets stored within one infrastructure and interlinked via the LinkSets. We give a more formal and detailed look at the data integration challenges for knowledge graphs in Chapter 5.

Physically integrated approach. The unified physically integrated approach implies a fusion of facts about an entity extracted from different data sources into one unified entity. Thus, the result is a physically new knowledge graph whose concepts are interlinked with the original datasources, e.g., via `owl:sameAs` or `owl:equivalentClass`. The overview represented in Fig. 4.3(a) shows that the unified knowledge graph is a stand-alone structure which consists of fused entities under a new URI. The structure connects the data from the remote datasources *both physically and logically*. The newly created entities are interlinked with the original sources via the `owl:sameAs` property. The approach is widely used by enterprises and the community, e.g., Google Knowledge Vault, Microsoft Satori (the knowledge graph behind Bing), DBpedia and Wikidata. There exist several methods to build an ontology for a unified EKG. The first method implies creating an ontology completely from scratch which results in a new set of classes and properties like a class `customURI:Book` and a property `customURI:author`. The second method relies on the utilization of concepts and roles of existing vocabularies, e.g. DBpedia terms `dbo:WrittenWork` and `dbo:author` respectively. The second case means that a certain schema assumes the role of an authoritative vocabulary. However, since chances are that there are separate vocabularies behind each unified KG, this also introduces the need to identify which of those vocabularies is the most authoritative for that domain. Practically, this means that one of the background schemas assumes the role of primary vocabulary, introducing also the need for a conversion process for data in other KGs that was originally described using the other schemas.

Virtually integrated approach. The virtually integrated approach employs a more distributed architecture based on remote sources and their access interfaces, e.g., a federation between DBpedia and Wikidata, including their SPARQL endpoints. The efforts are put into the process of interlinking schemas

Table 4.2: Comparison of EKG features

Criteria	**Physical**	**Transitionary**	**Virtual**
A: Physical storage and processing			
# of datasources	*n+1*	*n*	*n*
Computational requirements	Transformation, federation and fusion	Maintenance of and support for sameAs links	
Reliability	Failure if the unified EKG is down	Highest	Depends on availability of sources
B: Semantic structuring			
Knowledge organization	One ontology allows various organization schemas	Constrained ontologies over constrained local replicas	Sophisticated reasoning over different ontologies
Resource identification	A new namespace	Both can be used	Re-usage of existing URIs
Update propagation and replication	Permanent data fusion is required	Maintain *n* sources	
Update frequency	Revision-like	Depends on local replicas updates	Updates automatically as remote datasources are updated
Named graphs	Can be used	Each source is a named graph	Can be used
C: Access and maintenance			
Querying	Big chunks of data from one source	Less portions of data from lots of sources	
Redundancy & Duplication	Optimized SPARQL responses	Regulated via the replication policy; data duplication	Redundant SPARQL responses; data duplication
Access control	Fast; manual split might be required	Slower but flexible due to multiple access interfaces	Constrained to means built in the remote datasets

and instances without creating a single physical EKG (cf. Fig. 4.3(c)). Instead, the entities from the distributed sources complement each other. The EKG is then a *logical* structure represented by mappings between the remote datasources, as shown in Fig. 4.3(c). The storage space contains *LinkSets* between the datasources which maintain necessary mappings. An essential task in the federated approach is therefore to create and maintain relevant `owl:sameAs` links.

Transitionary approach The "golden mean" between the physical and virtual approaches aims at incorporating advantages of both. As shown in Fig. 4.3(b), the transitionary approach relies on local replicas of remote datasources stored as named graphs. On the one hand the data is stored in one place similar to the unified method, on the other hand an EKG still consists of LinkSets similar to the federated method. The transitionary approach allows for better reliability in comparison with the federated datasets on the Web and higher performance typical for the unified method. The approach can also be names as *semantically federated* or *loosely unified*.

4.1.4 EKG Features

This section contains a description of EKG technical features presented in Table 4.2. The features are grouped into three dimensions, i.e., physical storage and processing, semantic structuring, access and maintenance. We compare the above data and knowledge integration approaches using these features.

Physical storage and processing

Number of KBs to maintain slightly differs. Assuming that the fusion of Wikidata, Freebase and DBpedia follows the virtual approach, the target EKG would consist of the original datasets and semantic links between their entities. In contrast, following the physical approach the target graph is comprised of the sourced transformed, e.g., via some *extraction-transform-load* procedures, to one format and consists of entities that link to points in some or all of the three datasets. The physical approach follows a *data warehouse* fashion where all sources are converted to one physical representation. Therefore, in this example the virtual approach operates across just three original graphs (generally, *N*) whereas the physical approach creates an additional one (*N+1*).

Computational requirements. In the virtual approach the main computational effort consists of the maintenance of semantic links, e.g., `owl:sameAs` links, i.e., to reflect created, updated or deleted triples in the target EKG. A changed fact needs to invoke processes that immediately adjust the current set of semantic links to the updated state. Query federation is the second consumer of computational power in this approach. Queries and the infrastructure should be optimized in a way that supports federated KBs, e.g., access to different machines, as well as semantically federated KBs, e.g. support for separate named graphs. We dive into more details of federated query processing in Chapter 6.

In the physical approach the process of interlinking takes place before writing triples in the target graph. Therefore, data transformation and fusion are highly dependent on the performance of the integration framework. In the first step, the local dumps of sources are updated. The data fusion operation is then invoked with the new data and the result is finally (re)written in the target graph. In Section 5.2 we introduce a notion of *data fusion policies* and analyze their applicability for data and knowledge integration despite a chosen approach.

Reliability is one of the most important aspects of enterprise services. In the federated virtual approach, federation is considered to be less stable and reliable because of the nature of Web services and the need to gather data from the distributes sources for real-time processing. If one data source is unavailable it cuts off all the knowledge from the logical EKG, significantly reducing the semantic integrity. Such a federation is thus less reliable given the potential instability of remote datasources.

Semantic federation, where the data from different data source is stored locally, provides higher reliability. If one or more source graphs is inaccessible, the production state of the framework is maintained via previous dumps or available Web sources. The federated architecture enables advantages of quick restoration for the target graph.

Working with the physical approach, the attention is drawn to two possible types of failure. Unavailability of one or more sources does not affect the target graph, given the availability of previously obtained data before the fusion process. The target graph is then updated when the sources are available again. In case of target graph failure the only remaining option is to conduct the linking again from scratch (which is explained by the nature of the unified approach) or rely on a previous data dump.

Semantic structuring

Knowledge organization significantly affects the chosen integration approach. Both virtual and physical approaches perform well on the knowledge graphs of a common organization, i.e., triples or quads with provenance information. However, the development of more sophisticated logics and reasoning mechanisms facilitates the emergence of new types of knowledge graphs [144]. Probabilistic graphs introduce an uncertainty dimension to the common knowledge organization structure [62, 145, 146]. Temporal graphs leverage Markov Logic to tackle the time dimension [147]. Within the virtual approach the costs of maintaining logical consistence and data fusion between graphs of such disparate knowledge organizations becomes dramatically high. It will require usage of hybrid logics to transform one logical representation of a concept into another. Such hybrid logics are at the moment highly tailored and not mature enough to be used in enterprises [148]. Nevertheless, once created such a mechanism would outperform separated physical sources in terms of data scalability and interlinking. This would be analogous to the way in which the Internet tremendously outperforms local databases. The physical approach allows the seamless fusion of data under a certain knowledge organization type, inheriting strengths and shortcomings of such a method. For the above reasons, an EKG is usually designed around only one of logic and reasoning mechanisms, e.g., a probabilistic EKG, a fuzzy logics-based EKG, a Markov Logic-based EKG, etc.

The logical difference in the knowledge organization is outlined in the Table 4.3 which contains the facts about the *Interstellar* movie. The first column *Physical EKG* represents the triple structure of the EKG built according to the unified physical approach with a custom ontology. The second column *DBpedia* and the third column *LinkedMDB* represent the data sources of the transitionary approach. The datasets of the transitionary EKG often duplicate the information as values of disparate properties, e.g., `rdfs:label` and `dc:title`. Several facts are contradictory, e.g., the property value of `movie:director/8477` in LinkedMDB leads to Steven Spielberg whereas the real director is Christopher Nolan. Some facts might be presented in one datasource but not presented in another, e.g. the "budget" and "gross" triples are available in DBpedia but are missing in LinkedMDB. The above-mentioned conflicts must be thoroughly examined and effectively resolved during the data fusion process. The physically integrated EKG contains the knowledge fused from the two data sources using some fusion policy and stores the triples under a custom URI. `owl:sameAs` links refer to original datasources the knowledge was extracted from.

The physical EKG aims at providing a universal description of an entity using the information from available data sources. Thus it raises the importance of data fusion as listed in the Table 4.2. Data sources often contain the same or overlapping information as values of disparate properties, e.g. the title of the film is attached as `rdfs:label` to a movie in DBpedia and as `dc:title` in LinkedMDB. Some facts do not match at all, e.g. the property value `movie:director/8477` of LinkedMDB leads to Steven Spielberg whereas the real director is Christopher Nolan. Some facts might be presented in one datasource but not presented in another, e.g. the "budget" and "gross" triples are available in DBpedia but are missing in LinkedMDB. The above-mentioned conflicts must be thoroughly examined and effectively resolved during the data fusion process. We try to tackle this challenge in Chapter 5 describing a knowledge graph integration framework that specifies principles and rules of solving fusion conflicts.

Resource identification. In the virtual approach semantic links, e.g., `owl:sameAs`, connect individuals from source graphs, e.g., Wikidata, Freebase and DBpedia using their URIs. Existing vocabularies with respective URIs are re-used so that there is no need to introduce a custom naming. For example,

```
dbpedia:XYZ  owl:sameAs  wikidata:ABC
```

In the physical integration approach a new graph is created and a custom URI is introduced:

```
customURI:ABC  owl:sameAs  dbpedia:XYZ
```

Table 4.3: Example of EKG data fusion approaches. A physically integrated graph contains fused facts under a newly defined namespace. A transitionary integration approach retains original graphs with their schemas.

Physically integrated graph, PREFIX `custom:`	**DBpedia**, PREFIX `dbo: dbp: dbr:`	**LinkedMDB** PREFIX `movie:`
`custom:Interstellar a custom:film`	`dbr:Interstellar_(film) a dbo:Film`	`movie:film/8304 a movie:film`
`rdfs:label "Interstellar"`	`rdfs:label "Interstellar"`	`dc:title "Interstellar"`
`custom:director custom:Christoper_Nolan`	`dbo:director dbr:Christopher_Nolan`	`movie:director movie:director/8477`
`custom:composer custom:Hans_Zimmer`	`dbo:musicComposer dbr:Hans_Zimmer`	`movie:music_contributor movie:music_contributor/4290`
`custom:writer custom:Jonathan_Nolan`	`dbo:writer dbr:Jonathan_Nolan`	`movie:writer movie:writer/14016, movie:writer/16958, movie:writer/18905`
`custom:producer custom:Emma_Thomas, custom:Lynda_Obst`	`dbo:producer dbr:Emma_Thomas, dbr:Lynda_Obst`	`movie:producer movie:producer/12918, movie:producer/9752`
`custom:$budget "165 000 000"`	`dbo:budget 1.65E8`	–
`custom:$gross "671 400 000"`	`dbo:gross 6.714E8`	–
`owl:sameAs dbr:Interstellar_(film)`	`owl:sameAs movie:film/8304`	`owl:sameAs dbr:Interstellar_(film)`
`owl:sameAs movie:film/8304`		

Custom URIs imply tackling the ontology alignment task to support consistent mapping between the remote datasources and a local schema.

Update propagation and Replication. Updates are vital for an EKG to allow a customer to receive the most recent and up-to-date information. The simplest form of handling updates is achieved in the physically federated EKG. The EKG is updated at the same moment as the remote datasource. The only action required is to adjust LinkSets according to the new facts. In the semantically federated approach the goal of the update of a certain datasource is to replicate it within the target KB. Therefore in case of three sources, three replications need to be maintained in parallel. The target KB is updated naturally as soon as all the new triples are processed by the linking engine.

For the physically integrated EKG a more sophisticated mechanism is required. A straightforward solution is to perform data fusion each time (or periodically) the remote datasources are updated. It evidently demands much computational resources as it requires permanent transformation, fusion and conflict resolution. Another solution is to use update propagation frameworks such as the one proposed by [149], which have the potential to automatically perform this process. However, the majority of such frameworks are not yet mature enough to be implemented in high-load enterprise settings.

Named Graphs play an essential role only in the semantically federated approach alternative. Each datasource has to be wrapped, for instance, in its own named graph, in order to make federated queries. On the other hand, in the physical approach the target graph contains all the information about semantic links and thus, named graphs are not mandatory and can be used up to a specific task.

Consistency. Both approaches provide high consistency. Virtual federated EKGs follow the principles of networking, URI re-usage and accessibility which results in a higher level of logical completeness of its content at the cost of logical duplicates. The physical KB obtains completeness and comprehensiveness during the data fusion stage containing refined and processed triples.

Access and maintenance

Querying. Queries are the main communication interface between the EKG and services over it. The two aspects of querying are Incoming queries (from services) and Outgoing queries (to the datasources).

The virtually integrated federated EKG processes only one type of queries from the clients and services. Considering the varying granularity of the data sources, the queries tend to be smaller in size but the number of queries depends on the number of data sources. The queries are executed over separate graphs via separate SPARQL endpoints.

In the physical approach a fused graph receives Incoming queries and sends Outgoing queries to its remote sources for additional information or to update local dumps of those sources. The size of received queries and sent responses tends to be larger, since the unified EKG attempts to contain as much information about a concept as possible. We study EKG federated query processing challenges in Chapter 6 and propose improvements on various conceptual levels in order to increase query performance.

Redundancy & Duplication. Virtually integrated EKGs tend to contain logically duplicated triples thus increasing redundancy and the size of SPARQL responses. In case of the semantic federation the remote datasources should be replicated in the local infrastructure thus increasing the duplication. Physical EKGs resolve the problem of redundancy during the fusion stage saving the refined information in the unified KB. However, update maintenance requires constant updating of the local replicas so the data duplication becomes noticeable. Redundancy, duplication, and other integration issues are considered during the knowledge graph integration pipeline presented in Chapter 5.

4.2 EKG Experimental Study

4.2.1 EKG Technologies

In this section, existing enterprise information systems (EISs) are analyzed and compared using a new unified assessment framework. We conduct an evaluation study, where cluster analysis allows for identifying and visualizing groups of EISs that share the same EKG features. More importantly, we put our observed results in perspective and provide evidences that existing approaches do not implement all the EKG features, being therefore, a challenge the development of these features in the next generation of EISs.

An Assessment Framework for Comparison

We propose an independent assessment framework to compare features and shortcomings of each surveyed technology. The benchmark categorizes high-level EKG functionality along three dimensions: D1) Human Interaction; D2) Machine Interaction; and D3) Strategical Development. Each dimension consists of a number of features. All the features of each dimension and their possible values are presented in Table 4.4. These features are later utilized in the review of EISs.

Human Interaction (HI) enables humans to interact with an EKG. Modeling expressivity characterizes the degree to support a user in modeling knowledge or a domain of discourse behind an EKG ranging from taxonomies and simple vocabularies to complex ontologies, axioms, and rules. The feature specifies a level of expressivity available to a user with a certain technology. Curation describes the support for creating, updating, and deleting knowledge from an EKG, and the availability of comprehensive user interfaces to perform such operations. Various aspects of collaborative (multi-user) editing, authorship and content curation are investigated. Surveyed EKG technologies offer different tools for data curation, i.e., graphical canvases on which a user can drag & drop graphical primitives to build an envisioned model, forms to fill in, text editor where a user writes certain statements in a certain syntax to embody a model. Linking comprises the level of support for establishing coherence between knowledge structures and instance data. Tools and techniques to connect domain-specific knowledge with other internal and external datasets provide high linking functionality. For instance, a domain-level materials vocabulary

Table 4.4: Benchmark Dimensions (D) – D1: Human Interaction; D2: Machine Interaction; and D3: Strategical Development. Benchmark Parameters (P) – P1: Modeling; P2: Curation; P3: Linking; P4: Exploration; P5: Search; P6: Data Model; P7: APIs; P8: Governance; P9: Security; P10: Quality & Maturity; P11: Provenance; and P12:Analytics. Dash – no feature

Dim.	#	Parameter	1	2	3	4	5	6
$D1$	$P1$	Modeling	Taxonomy	Thesaurus	Meta-schema	(SKOS) Vocabularies	Ontologies	Rules
	$P2$	Curation	Collaborative	plain text	forms	GUI	Excel	–
	$P3$	Linking	Text Mining	LOD Datasets	Ontology Mapping	Manual Mapping	Spreadsheet Linking	Taxo-nomy
	$P4$	Exploration/ Visualization	Charts	Maps	Web GUI	Faceted browser	Vis Widgets	Graphs
	$P5$	Search	Semantic NL	Faceted	Federated Faceted	Full text semantic	Semantic search	–
$D2$	$P6$	Data Model	RDF	RDF + docs	RDBMS	Property Graph	Taxonomy	
	$P7$	APIs	SPARQL	REST	SQL	custom APIs	ESB & OSGi	Java/ Python
$D3$	$P8$	Governance	Policies	Best practices	SAP	RDF based	–	–
	$P9$	Security	Spring	ACL	SAP	Roles	Token	–
	$P10$	Quality & Maturity	SKOS based	RDF based	SAP	–	–	–
	$P11$	Provenance	Context	Data Lake based	SAP	–	–	–
	$P12$	Analytics	Statistical	Exploratory	SAP	NLP	Big Data	–

used in manufacturing could be linked with publicly available descriptions in DBpedia or Wikidata, or a list of business partners could be linked with their company descriptions in a public company register, and their financial KPIs to a financial KB. The linking technique might vary from the EKG implementation. Fully-automatic linking requires from a user only datasets to connect. The rest of the work is done automatically so that the result is a fully integrated dataset. Semi-automatic linking requires additional input from a user on a schema level or instance level. On the schema level a user manually creates mappings between certain parts of vocabularies whereas on the instance level the mappings are set between particular instances of vocabularies. Manual linking implies manual matching of particular entities. The mappings might be established via curation tools (GUIs, forms, plain text).

Exploration & Visualization functions are essential for a successful user experience. Simple, yet powerful interfaces facilitate appraisal and adoption of new technologies within the enterprise. Among lay users (here referring to non-specialists in semantic technologies), an ability to explore data easily and represent it in a desired way often determines a solution's usability and success. An EKG solution should thus provide a set of visualization techniques suitable for non-experts. Those might include interactive visualized graphs, faceted browsers or interactive SPARQL interfaces. Enterprise Search can be extended to allow for a semantic search over the entire enterprise information space, and beyond. Robust search over a company's products, a competitor's products, or various datasets significantly accelerates the flexibility and performance of medium and large enterprises where information flows are often tangled and disparate. EKGs can employ fast-evolving Question Answering systems based on machine learning [150] and semantic technologies [151] to tackle the analytical challenge.

The *Machine Interaction (MI)* dimension describes different levels of support for machine interaction with an EKG. As information systems of the next generation, EKGs accentuate machine-to-machine interaction by creating a shared information space that is understandable by automated agents (e.g., services, software, other information systems). The Data Models feature specifies the foundational model of an EKG, i.e., how the knowledge is stored and represented in memory. We distinguish between Document models, Relational DBs, Taxonomies [152], and the RDF data model. The Document model is the weakest knowledge organization model as data is stored in sets of documents of various formats. Such sets are often unordered, disparate and likely to contain contradictions in data. RDB models use relational databases to represent and store data. Taxonomy models organize data in hierarchical structures like trees. They define abstract metadata as an extra layer whereas the data itself is stored in its original format. As the most sophisticated data organization type, the RDF model allows EKGs to fully exploit benefits of semantic technologies. RDF Data is represented as triples and stored in graph databases. Whereas Data Models define the ‘depth’ of machine communication, available APIs expose the ‘breadth’ of the communication. An EKG is by definition involved in and connected with other information systems operating within an enterprise. APIs are a cornerstone and outline the difference between EKGs and previous generation of knowledge management solutions.

The *Strategical Development (SD)* dimension is orthogonal to the HI and MI dimensions. Strategical Development features are of equal importance for both dimensions. Governance indicates availability of management mechanisms within a particular EKG implementation. Rigorous policies of data exchange within an EKG facilitate high knowledge consistency and quality. EKGs for large enterprises should obey corporate policies and should be embedded into the decision-making routine. Thus a certain responsible committee determines strategical goals for an EKG, identifies important KPIs, produces policies and standards of interaction with the EKG and monitors the overall performance. A certain department has to implement and maintain an EKG while certain people are allowed to introduce changes to the knowledge graph. Governance principles should be a reference point for knowledge curation and EKG APIs. The Security feature is of high importance in large enterprises. EKGs should employ complex and comprehensive access control procedures, manage rights, and permissions on a large scale. Security might be realized on different levels of the EKG organization: on the vocabulary level by using dedicated properties for selected entities, on the database level by different ’views’ or excerpts prepared according to user permissions, on the API level by routing requests to allowed sources.

The Quality & Maturity (Q&M) feature enables corporate management to evaluate the quality of an EKG and track its evolution over time. Data Quality indicates a degree of compliance of EKG data to an accepted enterprise standard level. Maturity shows a degree of applicability of a certain technology in an enterprise. EKG maturity takes into account quality and functionality of EKG components. An EKG technology should have means to compute current quality and maturity indicators as to a specific part of an EKG like data consistency, availability of APIs. A consolidated score shows a current maturity level and problems to solve towards the next level. The Provenance feature allows for tracking origins from which the data has been extracted. Being applied to EKGs provenance elaborates on versioning and history of the EKG content. Knowledge provenance is crucial for decision support for the top management. Heavy-duty knowledge shared with partners in data value chains benefits from transparent provenance as well. Provenance might be extracted from metadata supplied with datasets, organized as named graphs in semantic triple or graph stores or as additional tables in the relational databases, implemented in version control systems. The Analytics feature provides statistics and overview of basic KPIs of an EKG. An analytical component takes advantage of the stored knowledge in order to provide information on certain business components, current status of an EKG, accessibility of EKG components, logs and journals. The analytical means among surveyed EKG technologies vary from comprehensive Tableau-like interactive dashboards leveraging exploratory analytics to Big Data analytics and statistical analysis which implement machine learning techniques. In addition, the Analytics dimension takes into account available services on top of the EKG. The variety of analytical services indicates aptitude of EKGs to be involved into enterprise information flows instead of being reluctant ’knowledge silos’.

Table 4.5: Benchmark-based Comparison of EKG Solutions. Parameter values are taken from Table 4.4: P1: Modeling; P2: Curation; P3: Linking; P4: Exploration; P5: Search; P6: Data Model; P7: APIs; P8: Governance; P9: Security; P10: Quality & Maturity; P11: Provenance; and P12:Analytics. Dash – no feature

System	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*	*P10*	*P11*	*P12*
Ontorion	5,6	1,2,3	3	1,2	1	1	1	–	–	–	–	1,4
PoolParty	1,2,5	4	1,4	3,5	2	1	1,2,3	–	1	1	–	1,2
KnowledgeStore	4	–	1,2	4	1	2	1,2	–	–	–	1	–
Metaphacts	5,6	1,2,3	1,2	4,5	1	1	1	–	–	–	–	1
Semaphore 4	4	4	1,4	4,5	3	1	1,2,4	1	–	–	–	2,4,5
Anzo SDP	5	2,4,5	1,5	3,5	4	3	1,4,5	2	2	–	–	2
RAVN ACE	3	–	1	3,5,6	4	4	2,4,6		2,5	–	–	2,4
CloudSpace	5	–	2	4,6	1	1	1	–	–	–	–	2
ETMS	1,4	3,4	6	6	–	5	3	1	4	–	–	–
SAP HANA	–	–	–	–	–	4	3	3	3	3	3	3
EKG	5,6	1,4	1,2,3	3,5	4	1	1,2,4,5,6	4	2,4,5	2	2	1,2,5

EKG Solutions

Below, we compare 10 of the most prominent available solutions that implement some of EKG functions to a certain extent. These are selected[2] based on their relevance given our own EKG definition, as well as their maturity (in a deployment-ready state and targeted for enterprise). The EKGs are evaluated against the benchmark presented in Section 4.2.1. Below we elaborate on the results summarised in Table 4.5.

Cognitum Ontorion[3] is a scalable distributed knowledge management system with rich controlled natural language (CNL) opportunities [153]. In the HI dimension, it enables sophisticated semantic modeling of high expressivity employing full support of OWL2 ontologies, semantic web rules, description logics (DL) and a reasoning machine. Collaborative ontology curation is available via a plain text editor and various forms. The platform supports ontology mapping out of the box although the type of mapping is not specified. Visualization and exploration means are presented by web forms and tables. CNL and semantic technologies stack enable natural language semantic search queries. In the MI dimension Ontorion employs RDF and SWRL to maintain the data model. Through the OWL API and SPARQL, machines can interact with the system. In the SD dimension Ontorion offers statistical analytics and Natural Language Processing (NLP) techniques.

Semantic Web Company PoolParty[4] is a software suite envisioned to enrich corporate data with semantic metadata in the form of a corporate thesaurus [154, 155]. The suite has a vast variety of functions which entirely cover the HI dimension. Knowledge can be modeled as a taxonomy, thesaurus, or as an ontology reusing built-in vocabularies, e.g., SKOS, schema.org, or custom. Curation is organized in the GUI via forms. Linking capabilities allow for text mining of unstructured data and manual vocabulary mapping. Internal linking processes unstructured data (text documents) exploiting text mining and entity extraction to link mined entities with existing ones or create new entries in the EKG. External linking leverages manual vocabulary mapping performed in the graphical editor. PoolParty provides web-based visualizations and exploration interfaces while traversing a taxonomy or ontology (visual browsing). SPARQL graphical shell is available to query the EKG. A wiki-based content browser might

[2]The KMWorld Magazine served as a major source for their identification. http://www.kmworld.com
[3]Web: http://www.cognitum.eu/semantics/Ontorion/
[4]Web: https://www.poolparty.biz/

also be used for data exploration. Semantic search supports facets, multilinguality, and structured and unstructured data including Sharepoint and Web CMS. To support MI, PoolParty uses the RDF model and implements JSON RESTful and SPARQL endpoints. In terms of SD features, Security relies on the Spring security mechanisms applied to the REST component of an EKG. Quality management is limited to SKOS vocabularies and checks potential errors and misuses. Analytical features perform statistical and exploratory data analysis.

KnowledgeStore[5] is a scalable platform [156] optimized for storing structured and unstructured data with the help of a predesigned ontology. KnowledgeStore is an integral part of the NewsReader[6] project. Modeling expressivity is limited to the predefined vocabulary so that newly acquired data must be represented according to this vocabulary. Linking functionality allows for automatic text mining and entity extraction with the subsequent juxtaposing the extracted data on the structured semantic data. External datasets like DBpedia and Freebase might be included into the knowledge graph for better entity recognition and knowledge enrichment. A SPARQL client and a faceted browser are responsible for knowledge exploration. KnowledgeStore might be integrated with Synerscope Marcato[7] for comprehensive multi-dimensional visualizations. Semantic search over a graph might be performed using natural language queries or entities URIs. The data model employs RDF in conjunction with unstructured documents. Resource layer stores unstructured data, e.g., documents in natural language, which metadata is described by the Dublin Core Terms vocabulary. Entity layer contains RDF triples which describe an entity and a context applied to a triple, e.g., a time frame during which a triple is true. Mention layer contains objects which help to refer an unstructured data snippet (resource) to a structured entity using a set of predefined properties. One can interact with KnowledgeStore via REST API or SPARQL endpoint. The described data model allows KnowledgeStore to leverage to an extent the Provenance feature. Data sources might be tracked on the instance level, a limited form of versioning is implemented by the triples context feature.

Metaphacts[8] is a platform to design, control, and visualize domain-specific KGs. As to the HI, Metaphacts supports a number of features. Modeling opportunities rely on OWL ontologies and reasoning mechanisms. Collaborative knowledge curation is available in the Web-based GUI via forms and plain text editor. Interlinking capabilities allow for semantic enrichment of data extracted from unstructured documents. External public datasets might be used for enrichment similarly to KnowledgeStore. Metaphacts offers visualization and exploration in the form of a faceted browser with custom visualization widgets. Semantic search queries are executed against structured RDF data in the EKG. For the MI dimension Metaphacts reuses open standards RDF and OWL having SPARQL endpoint as a communication interface. However, a distinctive feature of the platform is in the analytical domain as it provides machine learning functions enhanced by the GUI and custom visualization engine.

Smartlogic Semaphore 4[9] is a content intelligence platform which provides a broad spectrum of functions for an enterprise. The entire functionality of Semaphore 4 is accessible in the web-based GUI via a browser. In the HI dimension Semaphore enables rich modeling and knowledge curation opportunities capable of describing business processes and business units. The modeling is performed using taxonomies, SKOS vocabularies, and ontologies. Declarative ontologies serve as a backbone for subsequent text mining and content linking. Custom external vocabularies might be uploaded and edited as well. Moreover, particular concepts from external vocabularies might be reused in new vocabularies which is comprehensively depicted in the UI. Vocabularies might be interlinked on the schema level

[5]Web: `https://knowledgestore.fbk.eu/`
[6]Web: `http://www.newsreader-project.eu/`
[7]Web: `http://www.synerscope.com/marcato`
[8]Web: `http://metaphacts.com/`
[9]Web: `http://www.smartlogic.com/what-we-do/products-overview/semaphore-4`

manually by a user. Internal linking has greater capabilities powered by text mining, classification engine, data annotation and tagging. Data from unstructured documents is extracted and aligned with the vocabularies. Visualization and exploration means provide eloquent insights on the EKG and its construction routine. A user can switch between a tree and a graph visualization to explore the schema level. Faceted browser enabled exploration of instance data. Text mining GUI guides a user from uploading a document to facts extraction. Semaphore 4 enables semantic search capable of answering federated queries and faceted queries. Search over text documents is supported as well. Elaborating on the MI Semaphore is oriented toward RDF data model. One of the important advantages for an enterprise is a set of integration interfaces with other enterprise-level solutions, e.g., Apache Solr, Oracle WebCenter Content. In the SD aspect, Semaphore allows for the creation of governance policies and user workflows to follow a certain policy. Powered by Big Data and NLP Semaphore offers a comprehensive analytical framework with rich visualizations.

Cambridge Semantics Anzo Smart Data Platform[10] is a data integration platform which allows structured and unstructured data to be applied in the corporate information flows. Anzo SDP consists of several modules where each implements a particular pipeline. The majority of modules are available as a web application. However, certain data curation and integration modules are implemented as standalone applications or as extensions to Excel. The Unstructured module concentrates on transformation of unstructured data (e.g., texts in natural language) in semantic data. The Smart Data Lake module consolidates data within one environment integrated with other IT systems. The Data Manager module is responsible for data integration from various sources, whereas the Analytics module performs different kinds of analyses over the KG. As regards HI, Anzo SDP's components allow for data curation, interlinking, rich visualizations, and advanced semantic search functionality. Modeling expressivity relies on fully-supported RDF and OWL ontologies. Curation is performed in a standalone Ontology Editor which exports the created ontology to the Smart Data Lake. In addition Anzo provides an extension to build ontologies directly in Excel. Linking functionality is inclined toward internal linking. SDP provides semi-automatic means for integration of Excel spreadsheets with ontologies with user involvement. The Unstructured module allows numerous features for data extraction from text documents employing text mining techniques. The extracted knowledge is then transformed into structured data aligned with an ontology. SDP offers a comprehensive web-based GUI with a wide range of visualization widgets. Semantic search is supported by the NLP engine and is capable of operating on top of structured and unstructured content simultaneously. Natural language search involves all the entities in the Data Lake, i.e., text documents, ontologies, instance data. In the scope of MI Anzo SDP implements the RDF model built on top of Apache Hadoop HDFS, Apache Spark, and ElasticSearch. The platform might be integrated with existing IT solutions via numerous supported interfaces, i.e., Enterprise Service Bus (ESB), OSGi, SPARQL, JDBC, and custom APIs. As SD, the platform includes Governance, Security, and Analytics features. Governance is supported via default methodologies and best practices intended to preserve a robust workflow on each hierarchy level. A security mechanism is based on access control lists (ACL) and roles separation. Data analytics opportunities offered by SDP include exploratory analytics via interactive dashboards, sentiment analysis of text data, and spreadsheet analytics.

RAVN Applied Cognitive Engine[11] is an enterprise software suite for complex processing of unstructured data, e.g., text documents. ACE consists of numerous modules responsible for data extraction, search, visualization, and analytics. RAVN Knowledge Graph is a part of ACE platform. KG underpins powerful search functionality and offers detailed visualizations. Being mostly a text mining tool, ACE modeling expressiveness adopts characteristics of the inner abstract schema. Yet the details of the

[10]Web: http://www.cambridgesemantics.com/technology/anzo-smart-data-platform
[11]Web: https://www.ravn.co.uk/technology/applied-cognitive-engine/

schema remain unclear, the schema is used to store knowledge in the built-in graph database. Linking opportunities are presented with a broad spectrum of text analysis functions available in ACE. Entities are recognized, put into the KG and linked with similar entities extracted from other documents. ACE provides a comprehensive web-based GUI to maintain the extraction and processing timeline. GUI presents interactive search, KG visualizations, provides widgets for the analytics module. Full-text semantic search in natural language is supported by Apache Solr and graph database. To support MI, ACE employs the property graph model on top of MongoDB and ApacheSolr[12]. The platform provides a range of REST, Java, and Python APIs to interact with enterprise applications. To support SD, ACE benefits from the comprehensive security subsystem and analytical functions. Security is based on token strings and access level. In this case each entity has a list of allowed tokens and a list of denied tokens. If a user token is contained in the allowed list, then her access level is determined by subparts of the token string. ACE analytical functions employ NLP to supply a user with sentiment analysis, expert locator, and exploratory visualizations.

SindiceTech CloudSpace Platform[13] provides a set of tools to build and maintain custom EKGs. CSP applies the *Extract - Explore - Transform - Load* workflow on top of four layers which comprise the architecture of the . The layers are namely a scalable distributed Cloud layer, a Big Data layer which uses Hadoop and Hive to process large volumes of data, a Knowledge layer which utilizes W3C Semantic Web standards and a DB layer which is responsible for storage. CSP supports OWL ontologies for data modeling with high expressivity. The platform does not provide visual tools for curation. Nevertheless, ontologies can be uploaded via APIs. Linking functions allow for ontology-based integration when heterogeneous data from different sources is transformed to RDF and mapped with one ontology. Pivot Browser enables relational faceted browsing for data exploration. Ontologies can be visualized as graphs. Apache Jena provides a SPARQL endpoint shell. Search functionality is powered by Semantic Information Retrieval Engine (SIREn) based on Apache Solr and ElasticSearch. In the MI dimension, CSP utilizes the RDF model with SPARQL as a basic API. SD features of the platform provide exploratory analytical services, e.g., modeling debugging and content overview.

Synaptica Enterprise Taxonomy Management Software[14] is a technology to manage corporate taxonomies, classifications and ontologies. ETMS provides a broad set of tools to create, update, map and maintain taxonomies which comprises the Synaptica Knowledge Management framework. ETMS modeling expressivity is limited to taxonomies and SKOS vocabularies. Relations among taxonomy or vocabulary entities are supported, yet constrained to hierarchical structures or limited SKOS properties. Curation means are based on a Web GUI which presents a tree-like interface with forms to fill in. Linking functionality is supported by taxonomy mappings. Mappings between taxonomies might be established automatically whereas mappings of vocabularies can be done manually via forms, and drag & drop in the GUI. Taxonomies and vocabularies are visualized as trees, hyperbolic graphs, hierarchies, area charts, and tree maps. For MI support, ETMS implements the Taxonomy model. Supported for export and import formats are CSV, XLS, XML, SKOS and RDF. The platform exposes RDBMS APIs to connect the system to the rest of the corporate IT environment. ETMS can be integrated with Sharepoint as well. In the SD dimension, ETMS allows for data governance and security. Default policies and workflows specify 12 possible roles with certain access permissions. A user is categorized in one of such groups and has access to the constrained part of the taxonomy with limited rights.

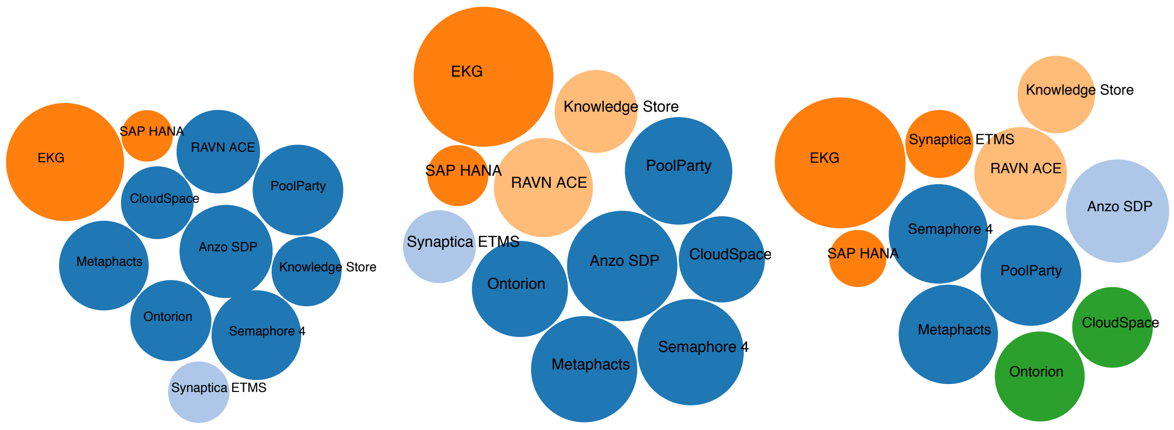
SAP HANA SPS 11[15] is a "one size fits all" enterprise software suite which includes an opportunity

[12]Hoeke,J.V.Ravn cor whitepaper: https://www.ravn.co.uk/wp-content/uploads/2014/07/CORE-Whitepaper-W.V1.pdf

[13]Web: http://www.sindicetech.com/cloudspace-platform.html

[14]Web: http://www.synaptica.com/products/

[15]Web: https://hana.sap.com/capabilities/sps-releases/sps11.html



(a) 3-clusters (b) 4-clusters (c) 5-clusters

Figure 4.4: **Clustering.** (a) Hierarchical clustering, n=3. The orange cluster offers rich SD functionality, the azure corresponds to the taxonomy management, the blue comprises RDF based systems; (b) EM algorithm, n=4. All of the above, but the nankeen cluster emphasizes text mining; (c) K-means algorithm, n=5. The green cluster employs link discovery tools.

to create and maintain EKGs on top of the HANA infrastructure and a specific graph database. HANA capabilities are overwhelmingly broad and cover all the features in the SD dimensions which are built-in in the default HANA distribution. However, an approach how HANA utilizes semantic technologies to support human interaction with an EKG remains vague. HANA Graph Engine is based on property graphs which naturally satisfy EKG needs. The engine exposes SQL-based APIs to communicate with versatile HANA modules and components.

EKG denotes the proposed EKG architecture with all the necessary features in HI, MI, and SD. This architecture employs knowledge graph integration framework described in Chapter 5 and federated query processing techniques described in Chapter 6.

Analysis of Existing Solutions

We conduct an unsupervised cluster analysis to existing EIS approaches for identifying groups of approaches of similar functionality[16]. Table 4.5 is transformed into a matrix with numerical values that denote normalized distances (in the range [0,1]) among the features, where values close to 1.0 indicate proximity to the defined EKG model. Values are chosen not to describe 'the best' or 'the worst' solution, but to stress the difference in functions the surveyed systems provide. *Weka*[17] is employed to perform the clustering into three, four, and five clusters. Varying the clusters number, we aim at identifying groups of EISs which share similar functions. One and two clusters give a superficial representation of the surveyed systems, and thus, are not included in the analysis. Three, four, and five clusters are able to reveal dependencies encoded in the vector representation which might have been overlooked or hidden during a manual review. Three clustering algorithms were applied, i.e., hierarchical algorithm, EM algorithm, K-means algorithm for three, four, and five clusters, respectively. We then visualized the results with *D3.js* visualization library. The radius of each node is proportional to the *Frobenius norm* of a features vector. The results are depicted in Fig. 4.4. The arrangement and positions of clusters are generated randomly, i.e., no cluster is logically closer to the nearest cluster than other clusters.

[16]Source codes are available in the github repository: `https://github.com/migalkin/ekgs_clustering`
[17]`http://www.cs.waikato.ac.nz/ml/weka/`

Fig. 4.4(a) represents the distribution in three clusters computed by the *hierarchical* clustering algorithm. The biggest `blue` cluster comprises systems which can be described as RDF-based systems with rich visualizations, semantic search and analytical functions. However, the blue cluster lacks the features of the *Strategical development* dimension. On the other hand, the `orange` cluster contains systems that implement features from this dimension. The size of the envisioned *EKG* approach is bigger due to the leverage of the *Human Interaction* features which SAP HANA SPS lacks. The smallest `azure` cluster contains only one system which distinguishes itself as a taxonomy management tool.

Fig. 4.4(b) represents a distribution in four clusters performed by the *EM algorithm*. The EM (Expectation-maximization) algorithm involves latent variables to maximize likelihood of the input parameters. The extent of divergence in features (*SD* and taxonomies, respectively) keeps the `orange` and `azure` clusters the same as in the previous case. However, a new `nankeen yellow` cluster is marked off from the `blue` cluster. The newly discovered cluster is characterized by the emphasis on text mining functions of the *HI* dimension. The `blue` cluster is still defined as semantic-based systems with rich visualizations.

Fig. 4.4(c) represents the distribution in five clusters calculated by the *K-means algorithm*. The algorithm partitions input data in clusters by comparing means of an input vector and the clusters. The input vector is attached to the nearest mean value cluster. Firstly, the `blue` cluster was split further. The systems in the `blue` cluster are characterized as RDF-based knowledge management solutions which rely on ontologies, collaborative visual editing and wide range of supported APIs. Secondly, a new `green` cluster, derived from the `blue` cluster, is described as a set of systems that is based on semantic technologies and is capable of performing data interlinking. Additionally, the `green` cluster exposes only SPARQL and OWL API communication interfaces. The `nankin yellow` cluster remained the same as a distinctive set of text mining solutions. The `orange` cluster added one system. Nevertheless, the cluster is still described as an enterprise-friendly collection of systems with features from the *SD* dimension. The `azure` cluster now comprises a different system, Anzo SDP. The cluster contains a system capable of spreadsheet processing using ontologies for high-level schema definition. The wide range of supported APIs and RDB implementation of RDF distinguish the `azure` cluster.

In the extensive evaluation section we investigate the effect of the chosen approach on the EKG performance along several dimensions, e.g., basic reasoning and OWL entailment which account for machine understanding of the EKG data, and access control subsystem which is of the utmost importance in large enterprises.

4.2.2 EKG Integration Approaches Analysis

In this paragraph, we place our emphasis on the computational resources required for instant querying, since this is often a bottleneck between the data consumer and the EKG. Therefore, we assess the query performance of the defined above integration approaches. The second focus of our attention is drawn to the access control issues and its effect on the EKG performance to be treated carefully in multi-user corporate environments. In order to address *RQ1* we break it down to smaller subquestions:

- What are the considerations, implications and practical expectations of different design strategies to realize an EKG?
- How do those strategies affect semantic complexity and decrease the query performance?
- What are the differences in the computational requirements?
- Is it possible to maintain low latency and permanent updates?

- What are the security mechanisms implementable in EKGs?

Comparing physical and semantically federated approaches

We evaluated the performance of the queries with simple reasoning to give an overview and predict the performance of EKGs built according to the unified physical integration and semantically federated (transitionary) approach. The selection is motivated by the same performance level of a system which ran the experiments, i.e., it does not depend on the performance of the remote datasources. The experimental configuration is as follows:

Benchmark: For our experiments we chose datasets in the movie domain from DBpedia and *LinkedMDB*[18]. The extracted datasets contain movies instances with all available properties. Table 4.6 presents an overview of the datasets used for the experiments.

Table 4.6: The datasources

	DBpedia	**LinkedMDB**
Size (MB)	1,160	891
Triples	6,400,756	6,147,996
– facts about movies	6,400,756	1,301,931
Films	89,941	85,620
Avg # of facts per film	71	15

Adhering to the transitionary approach the datasets were loaded into *Virtuoso 7.10.3207*[19] under the named graphs `http://dbpedia.org` and `http://linkedmdb.org`, respectively. We chose the DBpedia dataset as an example of the unified KG with already established `owl:sameAs` links among LinkedMDB and other datasets. The queries against the unified EKG are executed against the `<http://dbpedia.org>` graph. The queries against the transitionary EKG are executed against the `<http://dbpedia.org>` and `<http://linkedmdb.org>` graphs. The following queries are furnished with the Virtuoso pragma keyword `DEFINE input:same-as "yes"` which leverages `owl:sameAs` connection between the entities in the transitionary EKG. Lines starting with (*) indicate additional query terms for the transitionary EKG.

Query1 extracts all the triples from the datasources.

```
(*) DEFINE input:same-as "yes"
SELECT ?s ?p ?o
FROM <http://dbpedia.org>
(*) FROM <http://linkedmdb.org>
WHERE { ?s ?p ?o . }
```

Listing 4.1: Query1

Query2 extracts the movies and their titles.

```
(*) DEFINE input:same-as "yes"
SELECT ?film ?label
FROM <http://dbpedia.org>
(*) FROM <http://linkedmdb.org>
```

[18] http://linkedmdb.org
[19] http://virtuoso.openlinksw.com/

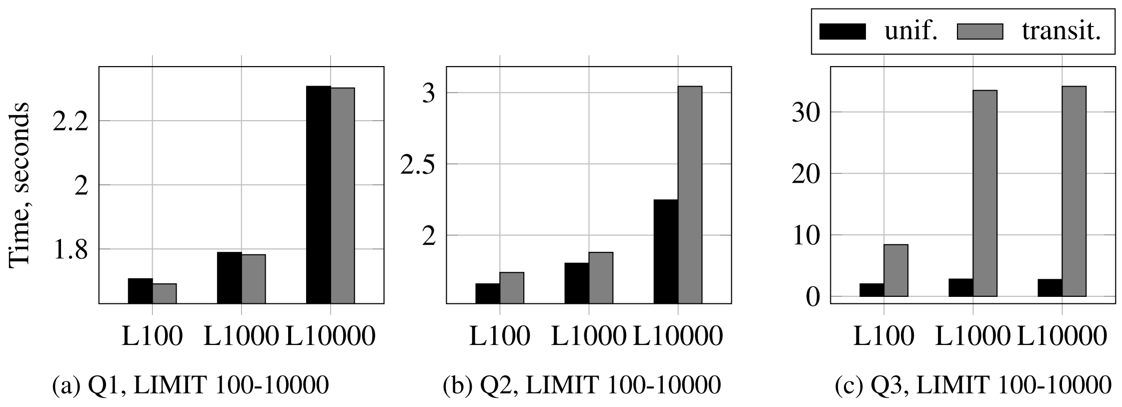


Figure 4.5: **Performance of the queries with simple reasoning, seconds**

```
WHERE {
        ?film a <http://dbpedia.org/ontology/Film> .
        ?film rdfs:label ?label . }
```

Listing 4.2: Query2

Query3 extracts all the entities which contain the string "graph" in the `rdfs:label` as well as their `owl:sameAs` values.

```
(*) DEFINE input:same-as "yes"
SELECT ?film ?same
FROM <http://dbpedia.org>
(*) FROM <http://linkedmdb.org>
WHERE {
        ?film rdfs:label ?label .
        ?film owl:sameAs ?same .
        FILTER (contains(str(?label),"graph")) }
```

Listing 4.3: Query3

Metrics: We report on execution time (ET in secs) as the elapsed time required by the SPARQL endpoint to produce all the answers of a query. We vary the amount of requested answers using the keyword `LIMIT` (`LIMIT 100`, `LIMIT 1000` and `LIMIT 10000`, respectively). In the second part we vary security configurations and the amount of named graphs.

Implementation: The queries were executed against Virtuoso in the HTTP mode to measure only the SPARQL performance independently of APIs peculiarities. The system used for the evaluation has a CPU Intel Core i5 2.4 GHz and 8 GB RAM.

The results are presented in Fig. 4.5. The queries were composed to utilize `owl:sameAs` reasoning available in Virtuoso via the keyword `DEFINE`. Q1 demonstrates almost no variation between the unified and the transitionary EKGs. The performance time was in an obvious correlation with the `LIMIT` on output values. Moreover, the performance of the unified EKG is almost constant in all the three cases. However, Q2 and Q3 indicate an increasing diversity in the query execution time. The execution time over the transitionary EKG raises dramatically in Q2 reaching a tremendous margin of more than one order of magnitude in the most complex Q3. That is, it takes more than 30 seconds to obtain 10,000 triples with the simplest entailment option enabled.

Table 4.7: Setup for the access control evaluation

Indicator	Measure
Triples	5,000,000
Named Graphs	500
Triples per named graph	10,000
Security groups	3
– Normal group	40% of named graphs
– Advanced group	30% of named graphs
– Confidential group	30% of named graphs
Users	Alice, Bob, Eve
– Normal group	Alice, Bob, Eve
– Advanced group	Alice, Bob
– Confidential group	Bob
Accessible graphs	
– Eve	40%
– Alice	70%
– Bob	100%

A number of factors produce a multiplied effect on the diverging performance of the semantically federated EKG. Sophisticated implementation of SPARQL entailment regimes in conjunction with triple store reasoning mechanisms result in the performance unacceptably low for a scalable reliable transitionary or federated EKG. We expect even lower SPARQL performance with the OWL2 RDF-based Semantics Entailment and OWL2 Direct Semantics Entailment regimes. The choice of using a physical or semantically federated approach, therefore, depends more on EKG goals, data volumes, and available computational resources. A physically integrated EKG tailored for a specific task is preferred when computational resources allow to transform source data into a new graph. This approach will benefit from better query performance on relatively simple queries. In situations where original data sources volumes are large enough and would consume unreasonable amount of time for a physical integration, a semantically federated approach is preferred. This choice will, however, require careful logical integration and mappings curation.

Impact of access control

In enterprise environments defining and enforcing access control policies to an EKG for different groups of employees and partners is of crucial importance. In order to assess the impact of access control regimes on the query performance, we ran a number of experiments aiming at *1)* determining the penalty introduced by having multiple security groups, *2)* determining the impact of the number of named graphs under an access control regime on the query performance. We chose the transitionary approach as the best approach suitable for the flexible access control rules modification. The system characteristics remained the same but the setup was changed as shown in the Table 4.7.

We created 500 named graphs with 10.000 triples in each graph. The triples were randomly extracted from the DBpedia SPARQL endpoint. Three security groups were created in Virtuoso, e.g. the Normal group accessible by everyone, the Advanced group accessible by two users and the Confidential group accessible only by one user. The allocation of named graphs by groups is presented in the Table 4.7, i.e. each 100 named graphs were allocated in three security groups. Approximately 40% of each 100 graphs are included in the Normal security group, 30% of the rest graphs are labeled as members of

Table 4.8: Experimental configuration

Group	**100 NGs**	**200 NGs**	**500 NGs**
Normal	40	80	200
Advanced	30	60	150
Confidential	30	60	150

(a) Named graphs (NGs) by a security group

User	**100 NGs**	**200 NGs**	**500 NGs**
Eve	40	80	200
Alice	70	140	350
Bob	100	200	500

(b) Named graphs available to a user

Table 4.10: Access control impact on the query execution time

	Time, seconds		
User	**100 NGs**	**200 NGs**	**500 NGs**
Eve	51.5	130.5	754.7
Alice	117.2	367.7	1842.2
Bob	205.9	1255.3	2854.5

the Advanced group and the last 30% belong to the Confidential group. The allocation for 100, 200 and 500 named graphs used in the experiment is presented in the Table 4.8(a). We then created three users, namely Alice, Bob and Eve, with various access rights and permissions. Eve can access the named graphs only from the Normal group, Alice can read the triples from both Normal and Advanced groups, Bob has the highest access level and can access all the graphs in three groups including the Confidential group. The distribution of named graphs available to a certain user is presented in the Table 4.8(b).

The following query was used to evaluate the query execution time depending on the user's access rights, i.e. how many named graphs are involved by Virtuoso to produce an answer. Virtuoso was configured to have an output limit of 100,000 triples per query. For this purpose the query has `LIMIT` and `OFFSET` keywords where `OFFSET` grows from 0 to the expected amount of triples, e.g. 1,000,000 for 100 named graphs queried by Bob.

```
SELECT *
FROM <http://ekgtest.com/Normal>
FROM <http://ekgtest.com/Advanced>
FROM <http://ekgtest.com/Confidential>
WHERE { ?s ?p ?o . }
LIMIT 100000 OFFSET x
```

Listing 4.4: Query4

Virtuoso access control mechanism protects named graphs if a user does not have a proper permission. That is, executing this query Eve will not have access to the Advanced and Confidential security groups despite the `FROM` keywords in the query. Alice will not access the named graphs in the Confidential group for the same reason.

We measured the query execution time observed by users with different access permissions for 100, 200 and 500 named graphs. The results are shown in Fig. 4.6, 4.7 and in Table 4.10. Each table entry is an average of three measurements rounded to one decimal place.

When analyzing the results, we observe that the complexity and the execution time as function of queried named graphs grow non-linearly. Security groups significantly affected the performance, i.e. a difference of more than one order of magnitude for all users whereas the amount of queried named graphs increased only in five times.

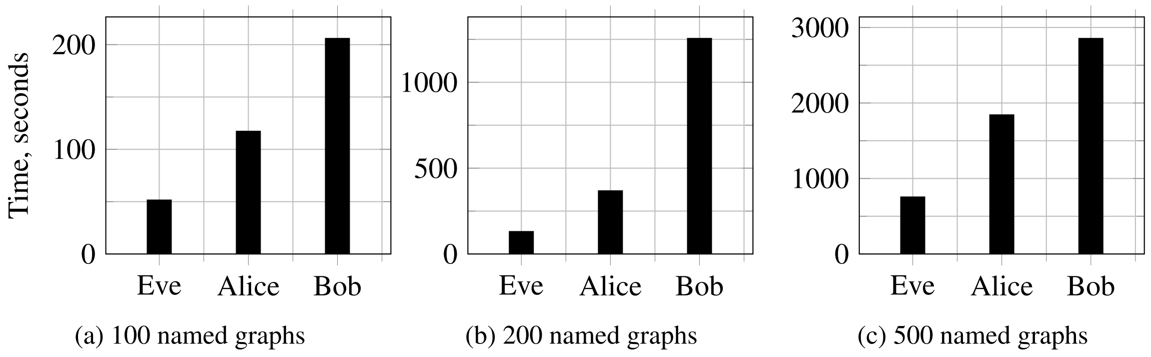


Figure 4.6: **Execution time of triples available to a user of a security group when num of NGs is fixed, seconds**

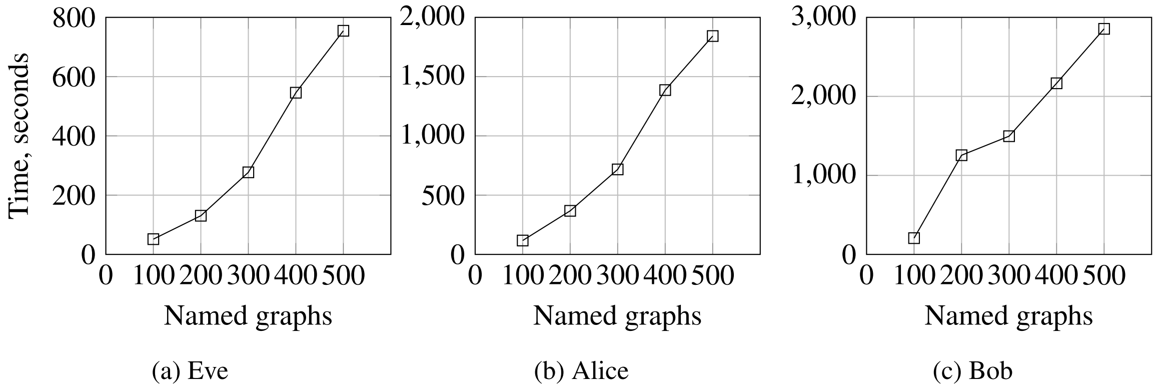


Figure 4.7: **Execution time within one fixed security group for NGs of various volume, seconds**

Bob's example shows that multiple security groups result in slower execution time though the number of involved named graphs might be the same. For example, it took almost twice as much time to process Bob's query involving three security groups and 200 named graphs (1255.3 seconds) as Eve's query involving one security group and the same 200 named graphs (754.7 seconds). Query performance within a fixed number of named graphs grows non-linearly, too. In all three cases the difference between the graphs available to Eve and Bob is 2.5 times (40 and 100, 80 and 200, 200 and 500, respectively) but the performance drops from by 3.8 (500 named graphs) to almost by 10 times (200 named graphs). We thus conclude that the Virtuoso security mechanism drastically exacerbates the query execution time especially in the EKGs populated by hundreds of small named graphs.

4.3 Summary

In this chapter, we presented the concept of an Enterprise Knowledge Graph, described its structure, functions, and purposes. Enterprise Knowledge Graphs are the next stage in the evolution of knowledge management systems. We discussed the most important requirements and features of EKGs which enterprises can select from to build a customized EKG. The physical, transitionary and virtual integration approaches expose their benefits in different cases. The unified physical integration approach has the advantage, that the enterprise has more control over the data and quality, and the data querying is significantly faster. Yet unified EKGs demand sophisticated data fusion mechanisms. In general, a virtual approach will be advantageous if the enterprise aims to continuously ingest updates and new additions from public LOD sources. Nevertheless, a certain overhead for query expansion and entailment regimes is required. The transitionary approach is advisable when data security plays a vital role. However, federated querying and access control over fine-grained datasources drastically impede the performance.

We devised an assessment framework to evaluate essential EKG properties along three dimensions related to the human interaction, machine interaction, and strategical development. We provided an extensive study of existing enterprise solutions, which implement EKG functionality to a certain extent. The analysis indicates a wide variety of approaches based on different data, governance, and distribution models and emphasizing the human and machine interaction domains differently. However, the strategical development functions, i.e., Provenance, Governance, Security, Quality, Maturity, which are of utmost importance for an enterprise, often still remain unaddressed by current EKG technologies. We see here a room for significant improvements and new innovative technologies empowering companies to fully leverage the EKG concept for data integration, analytics, and the establishment of data value chains with partners, customers and suppliers. We also report a practical study of the execution time of various reasoning-heavy queries and the effect of the access control modes on the query performance.

Experimental results provide enough empirical evidence to answer **RQ1** that graph-based models, i.e., RDF, are indeed able to represent actionable knowledge. Inherently supporting semi-structuring data, RDF knowledge graphs back up the concept of Enterprise Knowledge Graphs that aims to be a *holistic* knowledge management framework suitable for enterprise needs. EKG enjoys all the advantages of common knowledge graphs, tools, environment built around RDF, and provides an additional enterprise dimension to fulfil various enterprise needs, e.g., knowledge governance, access separation to valuable knowledge, continuous enrichment. In the following chapters we concentrate on particular knowledge graph integration and query processing frameworks that can be employed by EKGs.

CHAPTER 5

Heterogeneous RDF Graphs Integration

Chapter 4 presented a concept of Enterprise Knowledge Graphs. However, simple transformation of raw data sources into RDF does not lead to a knowledge graph creation. Newly obtained RDF graphs have to be thoroughly integrated, interlinked, and enriched with semantics in order to provide a uniform *black-box* view on the initial datasources. This Chapter is dedicated to Level 2 and Level 3 of the problem space (cf. Fig. 5.1), i.e., knowledge acquisition, integration and fusion. Once those those problems are solved, the resulting knowledge model can be considered as an instance of an EKG. Therefore, we study how heterogeneous RDF graphs, i.e., graphs with different schemas, can be semantically integrated. The following research question is addressed in this Chapter:

Research Question 2 (RQ2)

How can existing non-semantic data sources be lifted and semantically integrated leveraging enterprise knowledge graphs?

The knowledge integration pipeline (cf. Fig 5.2) defines three processing stages towards a consistent knowledge graph: 1) *Knowledge Acquisition* stage caters for raw data transformation into RDF using a certain ontology and mappings between the ontology and raw datasources (e.g., LAV paradigm); 2) *Knowledge integration* stage identifies similar entities among previously constructed RDF graphs using semantic similarity mechanisms; 3) Finally, during the *Interlinking* stage RDF graphs including

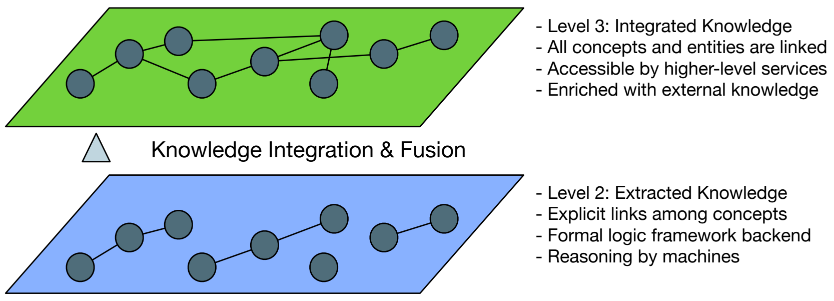

- Level 3: Integrated Knowledge
- All concepts and entities are linked
- Accessible by higher-level services
- Enriched with external knowledge
Knowledge Integration & Fusion
- Level 2: Extracted Knowledge
- Explicit links among concepts
- Formal logic framework backend
- Reasoning by machines

Figure 5.1: **The levels of the research problem addressed in this Chapter.** We concentrate on knowledge extraction, integration, and fusion tasks in order to achieve a unified view on the sources of a knowledge graph (i.e., Level 3).

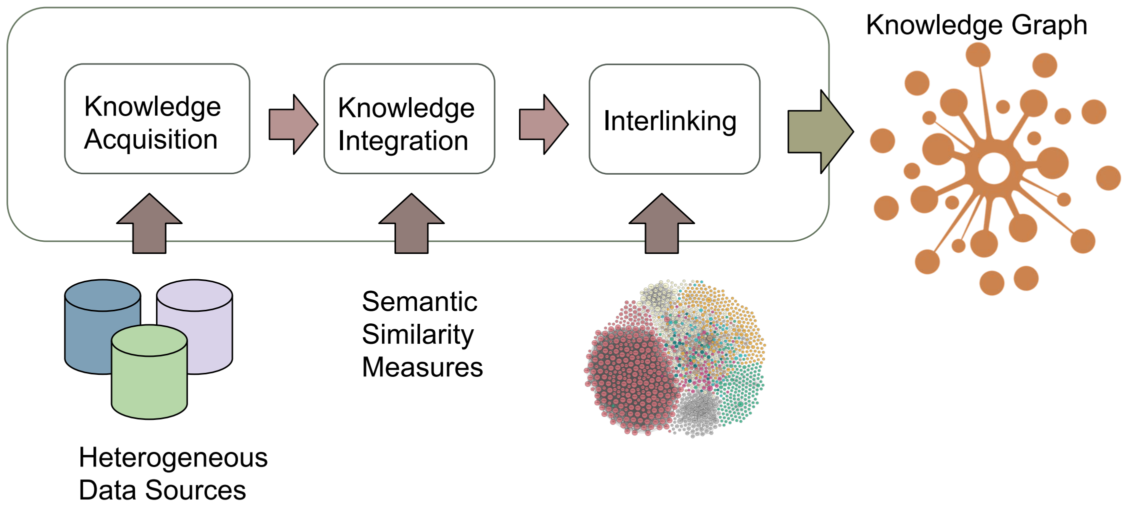


Figure 5.2: **Knowledge integration pipeline.** Knowledge acquisition stage transforms raw data into RDF graphs, Knowledge Integration stage identifies similar entities among the processed RDF graphs, and Interlinking stage performs fusion and outputs an integrated knowledge graph.

their similar entities are fused into one knowledge graph.

All the described stages leverage semantics encoded directly in the input datasources. In this Chapter, we focus on those three stages, i.e., *knowledge acquisition*, *knowledge integration*, and *interlinking*, as they still constitute a major problem on which results the knowledge graph heavily depends on. Approaching **RQ2**, we propose two methodologies for supporting the knowledge integration pipeline. The first framework caters for extracting knowledge from tables and tabular sources in general since tables are actively used for data representation in enterpises. That is, this approach covers the acquisition step of the pipeline. The second framework defines a physical operator that leverages a semantic data integration approach in a background and acts in a uniform join operator-like fashion suitable for embedding into query engines to produce integrated RDF graphs on the fly. That is, the operator covers integration and interlinking stages.

The stated **RQ2** benefits from the following contributions:

- A novel semantic methodology for acquiring knowledge from Web tables.
- *SJoin*, a semantic similarity join operator for producing integrated RDF graphs during query processing that adopts concepts of *RDF Molecules*, *fusion policies*, and *similarity functions*.

The following publications [157–160][1] describe the ideas the Chapter 5 is based on.

[1][158] is a joint work with Diego Collarana, another PhD student at the University of Bonn. In this paper, I contributed to the development of the semantic data integration pipeline, preparation of datasets for experiments, evaluation and analysis of obtained results, and reviewing related work.
[159] is a joint work with Diego Collarana, another PhD student at the University of Bonn. In this paper, my contributions include preparing a motivating example, contributions to a overall problem definition, formalization of fusion policies, complexity analysis, preparation of datasets for experiments, and reviewing related work.
[160] is a joint work with Diego Collarana, another PhD student at the University of Bonn. In this paper, I contributed to the definition and implementation of the semantic join operator, problem formalization and complexity analysis, preparation of datasets for experiments, evaluation and analysis of obtained results, and reviewing related work.

Following the semantic data integration pipeline, the Chapter is structured as follows. Section 5.1 discusses knowledge extraction techniques from tabular sources as tables still remain one of major sources on enterprise data. Section 5.2 provides necessary background understanding of a semantic approach for integrating heterogeneous RDF graphs that employs semantic similarity functions and *fusion policies*. We also adopt a notion of *RDF Molecules* that we use throughout this and following Chapters. Section 5.3 is dedicated to the description of *SJoin*, a semantic similarity join operator that is able to produce integrated RDF graphs *ad hoc* during query processing. In Section 5.4 we report results of the experimental study of the described techniques. Finally, in Section 5.5 we provide a summary of the achieved results and whether we can answer the **RQ2**.

5.1 Extracting Knowledge From Web Tables

The Web contains various types of content, e.g. text, pictures, video, audio as well as tables. Tables are used everywhere in the Web to represent statistical data, sports results, music data and arbitrary lists of parameters. Recent research [161, 162] conducted on the *Common Crawl* census[2] indicated that an average Web page contains at least nine tables. In this research about 12 billion tables were extracted from a billion of HTML pages, which demonstrates the popularity of this type of data representation. Tables are a natural way how people interact with structured data and can provide a comprehensive overview of large amounts and complex information. The prevailing part of structured information on the Web is stored in tables. Nevertheless, we argue that table is still a neglected content type regarding processing, extraction and annotation tools.

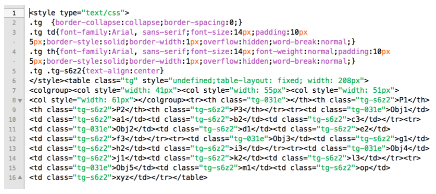
For example, even though there are billions of tables on the Web search engines are still not able to index them in a way that facilitates data retrieval. The annotation and retrieval of pictures, video and audio data is meanwhile well supported, whereas on of the most widespread content types is still not sufficiently supported. Assumption that an average table contains on average 50 facts it is possible to extract more than 600 billion facts taking into account only the 12 billion sample tables found in the Common Crawl. This is already *six* times more than the whole *Linked Open Data Cloud*[3]. Moreover, despite a shift towards semantic annotation (e.g. via RDFa) there will always be plain tables abundantly available on the Web. With this section, we want turn a spotlight on the importance of tables processing and knowledge extraction from tables on the Web.

The problem of deriving knowledge from tables embedded in an HTML page is a challenging research task. In order to enable machines to understand the meaning of data in a table we have to address the following questions: 1) Search for relevant Web pages to be processed; 2) Extraction of the information to work with; 3) Determining relevance of the table; 4) Revealing the structure of the table; 5) Identification of the table data range; 6) Mapping the extracted results to existing vocabularies and ontologies. The difference in recognizing a simple table by a human and a machine is depicted in Fig. 5.3. Machines are not easily able to derive formal knowledge about the content of the table.

The section describes current methodologies and services to tackle some crucial Web table processing challenges and introduces a new approach of table data processing which combines advantages of Semantic Web technologies with robust machine learning algorithms. Our approach allows machines to distinguish certain types of tables (genuine tables), recognize their structure (orientation check) and dynamically link the content with already known sources.

[2]Web: http://commoncrawl.org/
[3]Web: http://stats.lod2.eu/

	P1	P2	P3
Obj1	a1	B2	C3
Obj2	D1	E2	F3
Obj3	G1	H2	I3
Obj4	J1	K2	L3
Obj5	MkS	NxA	xyz



```
<style type="text/css">
.tg  {border-collapse:collapse;border-spacing:0;}
.tg td{font-family:Arial, sans-serif;font-size:14px;padding:10px
5px;border-style:solid;border-width:1px;overflow:hidden;word-break:normal;}
.tg th{font-family:Arial, sans-serif;font-size:14px;font-weight:normal;padding:10px
5px;border-style:solid;border-width:1px;overflow:hidden;word-break:normal;}
.tg .tg-s6z2{text-align:center}
</style><table class="tg" style="undefined;table-layout: fixed; width: 208px">
<colgroup><col style="width: 41px"><col style="width: 55px"><col style="width: 51px">
<col style="width: 61px"></colgroup><tr><th class="tg-031e"></th><th class="tg-s6z2">P1</th>
<th class="tg-s6z2">P2</th><th class="tg-s6z2">P3</th></tr><tr><td class="tg-031e">Obj1</td>
<td class="tg-s6z2">a1</td><td class="tg-s6z2">b2</td><td class="tg-s6z2">c3</td></tr><tr>
<td class="tg-031e">Obj2</td><td class="tg-s6z2">d1</td><td class="tg-s6z2">e2</td>
<td class="tg-s6z2">f3</td></tr><tr><td class="tg-031e">Obj3</td><td class="tg-s6z2">g1</td>
<td class="tg-s6z2">h2</td><td class="tg-s6z2">i3</td></tr><tr><td class="tg-031e">Obj4</td>
<td class="tg-s6z2">j1</td><td class="tg-s6z2">k2</td><td class="tg-s6z2">l3</td></tr><tr>
<td class="tg-031e">Obj5</td><td class="tg-s6z2">m1</td><td class="tg-s6z2">op</td>
<td class="tg-s6z2">xyz</td></tr></table>
```

Figure 5.3: **Different representations of one table.** On the left side a table follows a structured graphical order. On the right side a table with its contents is encoded in HTML format.

5.1.1 Formal Definitions

The foundation of the formal approach is based on ideas of Ermilov et al. [163]

A Table

Definition 5.1.1 *A table $\mathcal{T} = (\mathcal{H}, \mathcal{N})$ is tuple consisting of a header $\mathcal{H}$ and data nodes $\mathcal{N}$, where:*

- *the header $\mathcal{H} = \{h_1, h_2, \ldots, h_n\}$ is an n-tuple of header elements h_i. We also assume that the set of headers might be optional, e.g. $\nexists\mathcal{T} \equiv \mathcal{N}$. If the set of headers exists, it might be either a row or a column.*
- *the data nodes $\mathcal{N} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1} & c_{n,2} & \cdots & c_{n,m} \end{pmatrix}$ are a (n,m) matrix consisting of n rows and m columns.*

Table Genuineness

Definition 5.1.2 *The genuineness of the table is a parameter which is computed via the function of grid nodes and headers, so that $G_T(\mathcal{N}, \mathcal{H}) = g_k \in \mathcal{G}, \mathcal{G} = \{true, flase\}$.*

Table Orientation

Definition 5.1.3 *The orientation of the table is a parameter which values are defined in the set $\mathcal{O} = \{horizontal, vertical\}$ if and only if $G_T(\mathcal{N}, \mathcal{H}) = true$. So that $\exists\mathcal{O}_\mathcal{T} \iff G_\mathcal{T} \equiv true$. The orientation* *of the table is computed via a function of grid nodes and headers, so that $O_T(\mathcal{N},\mathcal{H}) = o_k \in \mathcal{O}$. If the orientation is horizontal, then the headers are presented as a row, so that $O_T \equiv$ horizontal. If the orientation is vertical, then the headers are presented as a column, so that $O_T \equiv$ vertical.*

Orientation Heuristics

Definition 5.1.4 *A set of heuristics $\mathcal{V}$ is a set of quantitative functions of grid nodes $\mathcal{N}$ and headers $\mathcal{H}$ which is used by machine learning algorithms in order to define genuineness and orientation of the given table $\mathcal{T}$.*

$$G_T(\mathcal{N},\mathcal{H}) = \begin{cases} true, & V_{g1} \in [v_{g1_{min}}, v_{g1_{max}}], \dots V_{gm} \\ false, & otherwise. \end{cases} \tag{5.1}$$

where $V_{g1}, ..., V_{gm} \in \mathcal{V}$, $[v_{g1_{min}}, v_{g1_{max}}]$ is the range of values of V_{g1} necessary for the true value in conjunction with $V_{g2}, ..., V_{gm}$ functions.

$$O_T(\mathcal{N},\mathcal{H}) = \begin{cases} horizontal, & V_{h1} \in [v_{h1_{min}}, v_{h1_{max}}], ..., V_{hn}. \\ vertical, & V_{v1} \in [v_{v1_{min}}, v_{v1_{max}}], ..., V_{vl}. \end{cases} \tag{5.2}$$

where $V_{h1}, ..., V_{hn} \in \mathcal{V}$, $V_{v1}, ..., V_{vl} \in \mathcal{V}$, $[v_{h1_{min}}, v_{h1_{max}}]$ is the range of values of V_{h1} necessary for the horizontal value in conjunction with $V_{h2}, ..., V_{hn}$ functions, $[v_{v1_{min}}, v_{v1_{max}}]$ is the range of values of V_{v1} necessary for the vertical value in conjunction with $V_{v2}, ..., V_{vl}$ functions.

Thus, it becomes obvious, that the *heuristics* are used in order to solve the classification problem [164] $X = \mathbb{R}^n, Y = \{-1, +1\}$ where the data sample is $X^l = (x_i, y_i)_{i=1}^l$ and the goal is to find the parameters $w \in \mathbb{R}^n, w_0 \in \mathbb{R}$ so that:

$$a(x, w) = sign(\langle x, w \rangle - w_o). \tag{5.3}$$

We define the terminological component $TBox_T$ as a set of concepts over the set of headers $\mathcal{H}$, and the assertion component $ABox_T$ as a set of facts over the set of grid nodes $\mathcal{N}$.

Hypothesis 1 *Tables in unstructured formats contain semantics.*

$$\exists \mathcal{T} | \{\mathcal{H}_T \iff TBox_T, \mathcal{N}_T \iff ABox_T\} \tag{5.4}$$

In other words, there are tables with the relevant content, which could be efficiently extracted as knowledge. We address the hypothesis proposing a framework for knowledge extraction from tabular sources. Presenting the experimental results in Section 5.4 we deduce whether the hypothesis holds true.

Algorithm 1 The knowledge extraction algorithms

```
 1: procedure SEMANTIFICATION
 2:     $URI \leftarrow$ specified by the user
 3:     tables localization
 4:     $n \leftarrow$ found tables
 5: while $n > 0$ :
 6:     $n \leftarrow n - 1$.
 7:     if genuine = true then
 8:         orientation check.
 9:         RDF transformation.
10:     goto while.
```

5.1.2 Extracting Knowledge from a Table

The knowledge extraction algorithm from tables is depicted as Algorithm 1. The valid URL of the website where the data is published is required from the user. The next step is a process of search and localization of HTML tables on a specified website. One of the essential points in the process is to handle DOM < *table* > leaf nodes in order to avoid nested tables. The following step is a computation of heuristics for every extracted table. Using a training set and heuristics a machine learning algorithm classifies the object into a genuine or a non-genuine group. The input of the machine learning module is a *table trace* – a unique multi-dimensional vector of computed values of the heuristics of a particular table. Using a training set the described classifiers decide, whether the vector satisfies the genuineness class requirements or not. If the vector is decided to be genuine the vector then is explored by classifiers again in attempt to define the orientation of the table. The correct orientation determination is essential for correct transformation of the table data to semantic formats. Then it becomes possible to divide data and metadata of a table and construct an ontology. If the table is decided to be a non-genuine then a user receives a message where it is stated that a particular table is not genuine according to the efficiency of a chosen machine learning method. However, the user is allowed to manually mark a table as a genuine which in turn modifies machine learning parameters.

Machine learning algorithms play a vital role in the system which workflow is based on appropriate heuristics which in turn are based on string similarity functions. The nature of the problem implies usage of mechanisms that analyze the content of the table and calculate a set of parameters. In our case the most suitable option is implementation of string similarity (or string distance) functions.

String Metrics. String metric is a metric that measures similarity or dissimilarity between two text strings for approximate string matching or comparison. *Levenshtein distance* is calculated as a minimum number of edit operations (insertions, substitutions, deletions) required to transform one string into another. Characters matches are not counted. *Jaro-Winkler distance* is a string similarity metric and improved version of the Jaro distance. It is widely used to search for duplicates in a text or a database. The numerical value of the distance lies between 0 and 1 which means two strings are more similar the closer the value to 1. Appraised by the industry [165], *n-gram* is a sequence of n items gathered from a text or a speech. In the paper n-grams are substrings of a particular string with the length n.

Metric Improvements. Due to the mechanism of tables extraction and analysis certain common practices are improved or omitted. Hence, we introduced particular changes in string similarity functions:

- The content type of the cell is more important than the content itself. Thus it is reasonable to equalize all numbers and count them as the same symbol. Nevertheless the order of magnitude of a number is still taken into account. For instance, the developed system recognizes 3.9 and 8.2 as the same symbols, but 223.1 and 46.5 would be different with short distance between these strings.

- Strings longer than three words have fixed similarity depending on a string distance function in spite of previously described priority reasons. Moreover, tables often contain fields like "description" or "details" that might contain a lot of text which eventually might make a mistake during the heuristics calculations. So that the appropriated changes have been implemented into algorithms.

Heuristics

Relying on the theory above it is now possible to construct a set of primary heuristics. The example table the heuristics mechanisms are explained with is:

Table 5.1: An example table

Name	City	Phone	e-mail
Ivanov I. I.	Berlin	1112233	ivanov@mail.de
Petrov P.P	Berlin	2223344	petrov@mail.de
Sidorov S. S.	Moscow	3334455	sidorov@ya.ru
Pupkin V.V.	Moscow	4445566	pupkinv@gmail.com

Maximum horizontal cell similarity. The attribute indicates the maximum similarity of a particular pair of cells normalized to all possible pairs in the row found within the whole table under the assumption of horizontal orientation of a table. It means the first row of a table is not taken into account because of a header of a table (see Table 5.2). Having in mind the example table the strings "Ivanov I.I." and "ivanov@mail.de" are more similar to each other than "Ivanov I.I." and "1112233".

The parameter is calculated under the certain rule:

$$maxSimHor = max_{i=2,n} \frac{\sum_{j_1=1}^{m} \sum_{j_2=1}^{m} dist(c_{i,j_1}, c_{i,j_2})}{m^2} \tag{5.5}$$

where i – a row, n – number of rows in a table, j – a column, m – number of columns in a table, *dist()* – a string similarity function, $c_{i,j}$ – a cell of a table.

Maximum vertical cell similarity. The attribute indicates the maximum similarity of a particular pair of cells normalized to all possible pairs in the column found within the whole table under the assumption of vertical orientation of a table. It means the first column of a table is not taken into account because in most cases it contains a header (see Table 5.3). According to the example table the parameter calculated for this table would be rather high because it contains pairs of cells with the same content (for instance "Berlin" and "Berlin").

Table 5.2: Horizontal orientation

H0	Header 1	Header 2	Header 3	Header 4
Obj 1				
Obj 2				
Obj 3				
Obj 4				

Table 5.3: Vertical orientation

H0	Obj 1	Obj 2	Obj 3	Obj 4
Header 1				
Header 2				
Header 3				
Header 4				

Using the same designations the parameter is calculated:

$$maxSimVert = max_{j=2,m} \frac{\sum_{i_1=1}^{n} \sum_{i_2=1}^{n} dist(c_{i_1,j}, c_{i_2,j})}{n^2} \tag{5.6}$$

It is obvious that the greater the maximum horizontal similarity the greater a chance that a table has vertical orientation. Indeed, if the distance between values in a row is rather low it might mean that those values are instances of a particular attribute. The hypothesis is also applicable to the maximum vertical similarity which indicates possible horizontal orientation.

Average horizontal cell similarity. The parameter indicates average similarity of the content of rows within the table under the assumption of horizontal orientation of a table. Again, the first row is not taken into account. The parameter is calculated under the certain rule:

$$avgSimHor = \frac{1}{n} \sum_{i=2}^{n} \frac{\sum_{j_1=1}^{m} \sum_{j_2=1}^{m} dist(c_{i,j_1}, c_{i,j_2})}{m^2} \tag{5.7}$$

where *i* – a row, *n* – number of rows in a table, *j* – a column, *m* – number of columns in a table, *dist()* – a string similarity function, *c*[*i,j*] – a cell of a table.

The main difference between maximum and average parameters is connected with size of a table. Average parameters give reliable results during the analysis of large tables whereas maximum parameters are applicable in case of small tables. **Average vertical cell similarity.** The parameter indicates average similarity of the content of columns within the table under the assumption of vertical orientation of a table. The first column is not taken into account.

$$avgSimVert = \frac{1}{m} \sum_{j=2}^{m} \frac{\sum_{i_1=1}^{n} \sum_{i_2=1}^{n} dist(c_{i_1,j}, c_{i_2,j})}{n^2} \tag{5.8}$$

Following Algorithm 1, the defined metrics and heuristics are employed by machine learning-based classifiers in order to identify genuineness of a table and its orientation. We evaluate the proposed approach paired with common machine learning algorithms, i.e., naive bayes, decision trees, support vector machines, and k-nearest neighbours, in Section 5.4.1. If a table is identified as genuine, a straightforward transformation to RDF is performed where each row becomes an individual and column names become predicates. On the other hand, hand-crafted mappings between column names and common ontologies can be utilized. Eventually, table contents are converted into distinct RDF graphs completing the knowledge acquisition task of the pipeline in Fig. 5.2. This section described how such graphs can be obtained from tabular data. Nevertheless, there exist numerous heterogeneous data sources employed by enterprises, e.g., Web APIs, unstructured text documents, databases, structured formats like CSV or JSON. In order to get a holistic view, two subsequent steps of the pipeline have to be taken, i.e., separated heterogeneous graphs have to be integrated and interlinked. That is, similar or equivalent graphs that refer to the same real-world entity have to be merged and those resulting unified graphs have to be enriched by semantic links between each other. In the following Section 5.2 we present background concepts that create a semantic data integration environment in which we define a physical operator to integrate and interlink heterogeneous RDF graphs described in Section 5.3.

5.2 Background: A Semantic Data Integration Framework

In this Section [4], we present a framework for semantic data integration as a necessary background that is leveraged in the following Section 5.3. We also adopt from [159] an essential definition of an *RDF Subject Molecule* and describe *fusion policies*.

RDF Subject Molecule [166]

Definition 5.2.1 *Given an RDF graph G, we call a subgraph M of G an RDF molecule [166] iff the RDF triples of $M = \{t_1, \ldots, t_n\}$ share the same subject, i.e., $\forall\, i, j \in \{1, \ldots, n\}$ $(subject(t_i) = subject(t_j))$. An RDF molecule can be represented as a tuple $\mathcal{M} = (R, T)$, where R corresponds to the URI (or blank node ID) of the molecule's subject, and T is a set of pairs $p = (prop, val)$ such that the triple $(R, prop, val)$ belongs to M. Property values are free of blank nodes, i.e., let I be a set of IRIs and L a set of literals, then $val \in I \cup L$.*

We call R and T the *head* and the *tail* of the RDF molecule $\mathcal{M}$, respectively. For example, an RDF molecule of the drug *Ibuprofen* is (`dbr:Ibuprofen`, {(`rdf:type`, `ChemicalSubstance`), (`dbo:actPrefix`, `"C01"`), (`pubchem`, `3673`)})[5]. Further, an RDF graph G is defined in terms of RDF molecules as follows:

$$\phi(G) = \{\mathcal{M} = (R, T) | t = (R, prop, val) \in G \land (prop, val) \in T\}$$

Given two RDF graphs G and D, an entity e can be represented by two different RDF molecules $\mathcal{M}_G$ and $\mathcal{M}_D$ in $\phi(G)$ and $\phi(D)$, i.e., $\mathcal{M}_G$ and $\mathcal{M}_D$ correspond to semantically equivalent RDF molecules as they both refer to one entity e. The proposed approach aims at identifying and merging such semantically equivalent RDF molecules from given RDF graphs.

The approach consists of two essential components. First, the identification component discovers semantically equivalent entities with the help of two sub-components, namely the Dataset Partitioner and the 1-1 Weighted Perfect Matching Calculator. Second, the Integrator component digests the output of the previous one in order to produce a semantically integrated knowledge graph.

Fig. 5.4 depicts the main architecture components. The pipeline receives two RDF graphs G and D, and additional parameters in order to produce a semantically integrated RDF graph. The whole approach relies on a semantic similarity measure Sim_f and an ontology O to determine when two RDF molecules are semantically equivalent. To do so, the Dataset Partitioner compares RDF molecules in $\phi(G)$ and $\phi(D)$ based on the similarity measure Sim_f. A bipartite graph is created between G and D; edges correspond to the pair-wise comparison of the RDF molecules and are weighted with values of the similarity measure Sim_f. A 1-1 weighted perfect matching algorithm is executed in order to identify for each RDF molecule the most similar one. Thus, if two RDF molecules are connected by an edge of the 1-1 perfect matching, then they are considered *semantically equivalent*.

Once the semantically equivalent RDF molecules have been identified, the second component of MINTE produces an integrated knowledge graph. In order to retain completeness and consistency

[4]This Section is based on [159], a joint work with Diego Collarana, another PhD student at the University of Bonn. In this paper, my contributions include preparing a motivating example, contributions to a overall problem definition, formalization of fusion policies, complexity analysis, preparation of datasets for experiments, and reviewing related work.

[5]We use standard prefixes according to `http://prefix.cc`

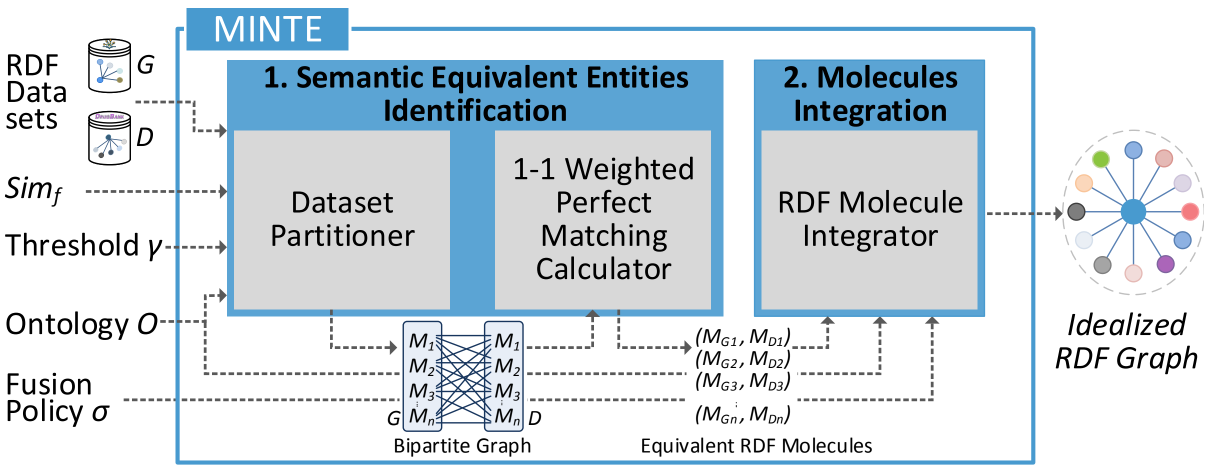


Figure 5.4: **Architecture of the framework.** The framework receives RDF datasets, a similarity function Sim_f, a threshold γ, an ontology O, and a fusion policy σ. The output is a semantically integrated RDF graph. A Dataset Partitioner creates a bipartite graph of RDF molecules and assigns the similarity value according to Sim_f, γ, and O. Semantically equivalent RDF molecules are related by edges in a 1-1 weighted perfect matching from the bipartite graph. Equivalent RDF molecules are integrated according to σ and mappings in O

and, at the same time, reducing the redundancy of the data, MINTE applies a set σ of *fusion policies*, i.e., rules operating on the triple level, which are triggered by a certain combination of predicates and objects. Fusion policies resemble flexible filters tailored for specific tasks, e.g., keep all literals with different language tags or discard all except one, replace one predicate with another, or simply merge all predicate-value pairs of given molecules. Fusion policies resort to an ontology O to resolve possible conflicts and inequalities on the levels of resources, predicates, objects and literals.

The framework defines three fusion policies, which are illustrated in Fig. 5.5:

Union policy. The union policy creates a set of $(prop, val)$ pairs where duplicate pairs, i.e., pairs that are syntactically the same, appear only once. In Fig. 5.5(a) the pair (p_1, A) is replicated, then it is included once in Fig. 5.5(b). The rest of the pairs are added directly.

Subproperty policy. The policy tracks if a property of one RDF molecule is an `rdfs:subPropertyOf` of a property of another RDF molecule, i.e.,

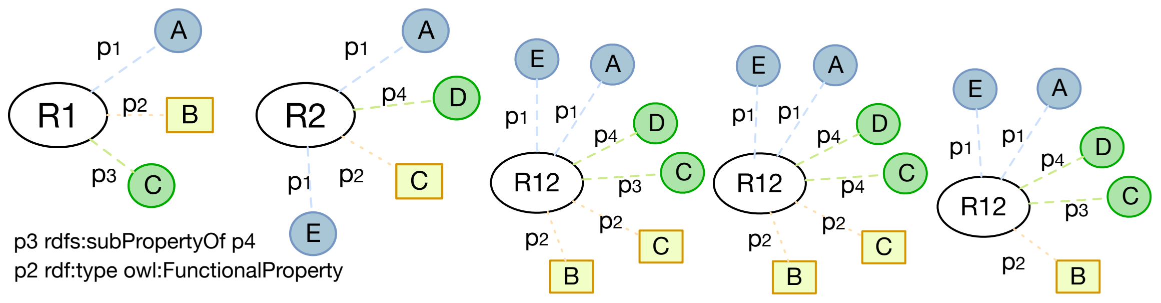
$$\{r_1, p_1, A\}, \{r_2, p_1, B\} + O + subPropertyOf(p_1, p_2) \models \{\sigma_r(r_1, r_2), p_2, \sigma_v(A, B)\}$$

. As a result of applying this policy, the property p_1 is replaced with a more general property p_2. The default σ_v object policy is to keep the property value of p_1 unless a custom policy is specified. In Fig. 5.5(c), a property p_3 is generalized to p_4 while preserving the value C according to the ontology axiom `p3 rdfs:subPropertyOf p4` in Fig. 5.5(a).

Authoritative graph policy. The policy allows for selecting one RDF graph as a prevalent source of data when integrating the following configurations of $(prop, val)$ pairs:

- The **functional property policy** keeps track of the properties annotated as `owl:FunctionalProperty`, i.e., such properties may have only one value. The authoritative graph policy then retains a value of a molecule from the primary graph:

$$\{r_1, p_1, B\}, \{r_2, p_1, C\} + O + functional(p_1) \models \{\sigma_r(r_1, r_2), p_1, \sigma_v(B, C)\}$$



(a) Semantically Equivalent RDF Molecules (b) Union (c) Subproperty (d) Functional

Figure 5.5: **Merging Semantically Equivalent RDF Molecules**. Applications of a fusion policy σ: (a) semantically equivalent molecules R_1 and R_2 with two ontology axioms; (b) simple union of all triples in R_1 and R_2 without tackling semantics; (c) p_3 is replaced as a subproperty of p_4; (d) p_2 is a functional property and R_1 belongs to the authoritative graph; therefore, literal C is discarded.

Annotated as a functional property in Fig. 5.5(a), p_2 has the value B in Fig. 5.5(d), as the first graph has been marked as authoritative beforehand. The value C is discarded. However, σ_v can redefine these criteria, and employ further processing to ensure that property values are equivalent or not.

- The **equivalent property policy** is triggered when two properties of two molecules are `owl:equivalentProperty`:

$$\{r_1, p_1, A\}, \{r_2, p_2, B\} + O + \mathit{equivalent}(p_1, p_2) \models \{\sigma_r(r_1, r_2), \sigma_p(p_1, p_2), \sigma_v(A, B)\}$$

  The authoritative policy selects a property from the authoritative graph, e.g., either p_1 or p_2. By default, the property value is taken from the chosen property. Custom σ_v policies may override these criteria.

- The **equivalent class or entity policy** contributes to the integration process when property values are annotated as `owl:equivalentClass` or `owl:sameAs`, i.e., two classes or individuals represent the same real-world entity, respectively:

$$\{r_1, p_1, A\}, \{r_2, p_2, B\} + O + \mathit{equivalent}(A, B) \models \{\sigma_r(r_1, r_2), \sigma_p(p_1, p_2), \sigma_v(A, B)\}$$

  . Similarly to the equivalent property case, the value with its corresponding property is chosen from the primary graph. Custom σ_p policies may handle the merging of properties.

The spectrum of possible fusion policies is not limited to the list described above. Fusion policies allow for a flexible management, and for a targeted control of creation of an integrated knowledge graph. These policies vary from naming convention for resources to a fine-grained tuning of desired parameters. Varying a set of applied policies, it is possible to focus on a certain integration aspect

5.3 The Semantic Join Operator

Semi-structured data models like RDF naturally allow for modeling the same real-world entity in various ways. To illustrate this, consider chemicals and drugs represented in the Drugbank and DBpedia knowledge graphs. Using different vocabularies, drugs are represented from different perspectives. DBpedia contains more general information, whereas Drugbank provides more domain-specific facts, e.g., the chemical composition and properties, pharmacology, and interactions with other drugs. Fig. 5.6 illustrates representations of two drugs in Drugbank and DBpedia. *Ibuprofen*, a drug for treating pain, inflammation and fever, and *Paracetamol*, a drug with analgesic, and antipyretic effects. Firstly, Drugbank Uniform Resource Identifiers (URIs) are textual IDs (e.g., `drugbank:DB00316`[6] corresponds to *Acetaminophen* and `drugbank:DB01050` to *Ibuprofen*. In contrast, DBpedia utilizes human-readable URIs (e.g., `dbr:Acetaminophen` and `dbr:Ibuprofen`) to identify drugs. Secondly, the same attributes are encoded differently with various property URIs, e.g., `chemicalIupacName`, `casRegistryNumber` in Drugbank, and `iupacName`, `casNumber` in DBpedia, respectively. Thirdly, some drugs might be linked to more than one analogue, e.g., Acetaminophen in Drugbank (`drugbank:DB00316`) corresponds to two DBpedia resources: `dbr:Paracetamol`, and `dbr:Acetaminophen`.

Traditional join operators, e.g., *Hash Join* [45] or *XJoin* [99], are not capable of joining those resources as neither URIs nor properties match syntactically. Similarity join operators [100–104] tackle this heterogeneity issue, but due to the same extent of inequality string and set similarity techniques are limited in deciding whether two RDF resources should be joined or not. Therefore, we identify the need of a semantic similarity join operator able to satisfy the following requirements: R1) Applicable to heterogeneous RDF knowledge graphs. R2) Able to identify joinable tuples leveraging semantic relatedness between RDF graphs. R3) Capable of performing perfect matching for one-to-one integration, and fuzzy conditional matching for integrating groups of N entities from one graph with M entities from another knowledge graph. R4) Support of a blocking operation mode for batch processing, and a non-blocking mode for on-demand real time cases whenever results are expected incrementally. We present *SJoin* – a semantic join operator which meets these requirements.

5.3.1 Problem Statement

We tackle the problem of identifying semantically equivalent RDF molecules from RDF graphs. The definition of an RDF molecule follows the one provided in Definition 5.2.1. Given an RDF graph G, we call a subgraph M of G an *RDF molecule* [166] iff the RDF triples of $M = \{t_1, \ldots, t_n\}$ share the same subject, i.e., $\forall\ i, j \in \{1, .., n\}\ (subject(t_i) = subject(t_j))$. An RDF molecule can be represented as a pair $\mathcal{M} = (R, T)$, where R corresponds to the URI (or blank node) of the molecule subject, and T is a set of pairs *p=(prop,val)* such that the triple *(R,prop,val)* belongs to M. We name R and T the head and the tail of the RDF molecule $\mathcal{M}$, respectively. For example, an RDF molecule of a drug *Paracetamol* is (`dbr:Paracetamol`, {(`rdfs:label`,`"Paracetamol@en"`), (`dbo:casNumber`,`"103-90-2"`),(`dbo:iupacName`,`"N-(4-hydroxyphenyl)ethanamide"`)}). An RDF graph G can be described in terms of its RDF molecules as follows:

$$\phi(G) = \{\mathcal{M} = (R, T) | t = (R, prop, val) \in G \text{ and } (prop, val) \in T\}$$

[6]Prefixes are as specified on http://prefix.cc/

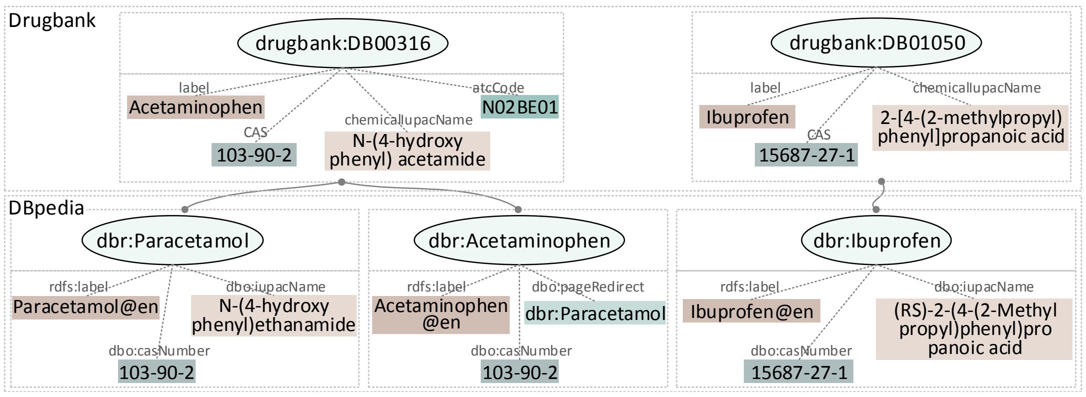


Figure 5.6: **Motivating Example.** The Ibuprofen and Paracetamol real-world entities are modeled in different ways by Drugbank and DBpedia. Syntactically the properties and objects are different, but semantically the represent the same drugs. Drug `drugbank:DB01050` matches 1-1 with `dbr:Ibuprofen`, while `drugbank:DB00316` matches 1-2 with `dbr:Paracetamol` and `dbr:Acetaminophen`.

Problem of Semantically Equivalent RDF Graphs

Definition 5.3.1 *Given sets of RDF molecules $\phi(G)$, $\phi(D)$, and $\phi(F)$, and an RDF molecule $\mathcal{M}_e$ in $\phi(F)$ which corresponds to an entity e represented by different RDF molecules $\mathcal{M}_G$ and $\mathcal{M}_D$ in $\phi(G)$ and $\phi(D)$, respectively. The problem of identifying semantically equivalent entities between sets of RDF molecules $\phi(G)$ and $\phi(D)$ consists of providing an homomorphism $\theta : \phi(G) \cup \phi(D) \to 2^{\phi(F)}$, such that if two RDF molecules $\mathcal{M}_G$ and $\mathcal{M}_D$ represent the RDF molecule $\mathcal{M}_e$, then $\mathcal{M}_e \in \theta(\mathcal{M}_G)$ and $\mathcal{M}_e \in \theta(\mathcal{M}_D)$; otherwise, $\theta(\mathcal{M}_G) \neq \theta(\mathcal{M}_D)$.*

Definition 5.3.1 considers perfect 1-1 matching, e.g., determining 1-1 semantic equivalences between `drugbank:01050` and `dbr:Ibuprofen`, as well as $N - M$ matching, e.g., `drugbank:DB00316` with both `dbr:Paracetamol` and `dbr:Acetaminophen`.

5.3.2 Proposed Solution: The SJoin Operator

We propose a similarity join operator named SJoin, able to identify joinable entities between RDF graphs, i.e., SJoin implements the homomorphism $\theta(.)$. SJoin is based on the *Resource Similarity Molecule (RSM)* structure, that in combination with a *similarity function* Sim_f, and a *threshold* γ, produce a list of matching entity pairs. RSM is defined as follows:

Resource Similarity Molecule (RSM)

Definition 5.3.2 *Given a set $\mathbb{M}$ of RDF molecules, a similarity function* Sim_f, *and a threshold γ. A Resource Similarity Molecule is a pair RSM=($\mathcal{M}$,T), where:*

- *$\mathcal{M} = (R, T)$ is the head of RSM and the RDF molecule described in RSM.*
- *T is the tail of RSM and represents an ordered list of RDF molecules $\mathcal{M}_i = (R_i, T_i)$. T meets the following conditions:*
  - *$\mathcal{M}$ is highly similar to $\mathcal{M}_i$, i.e., $\mathrm{Sim}_f(R, R_i) \geq \gamma$.*
  - *For all $\mathcal{M}_i = (R_i, T_i) \in T$, $\mathrm{Sim}_f(R, R_i) \geq \mathrm{Sim}_f(R, R_{i+1})$.*

An RSM is composed of a head and tail that correspond to an RDF molecule and a list of molecules which similarity score is higher than a specified threshold γ, respectively. For example, an RSM of *Ibuprofen* (with omitted tails of *property:value* pairs) is `((dbr:Ibuprofen,` T`)[(drugbank:DB01050,` T_1`), (chebi:5855,` T_2`), (wikidata:Q186969,` T_3`)])` given a similarity function Sim_f, a threshold γ, and Sim_f(`dbr:Ibuprofen,drugbank:DB01050`)$\geq$ Sim_f(`dbr:Ibuprofen,chebi:5855`), and Sim_f(`dbr:Ibuprofen,chebi:5855`)$\geq$ Sim_f(`dbr:Ibuprofen,wikidata:Q186969`).

The SJoin operator is a two-fold algorithm that performs: first, *Similarity Partitioning*, and second, *Similarity Probing* to identify semantically equivalent RDF molecules. To address batch and real-time processing scenarios, we present two implementations of SJoin. **Blocking SJoin Operator** solves the 1-1 weighted perfect matching problem allowing for a batch processing of the graphs. **Non-Blocking SJoin Operator** employs fuzzy conditional matching for identifying communities of *N-M* entities in graphs covering the on-demand case whenever results are expected to be produced incrementally.

5.3.3 Blocking SJoin Operator

Fig. 5.7 illustrates the intuition behind the blocking SJoin operator. Similarity Partitioning and Probing steps are executed sequentially. Thus, blocking SJoin operator completely evaluates both datasets of RDF molecules in the Partitioning step, and then fires the Probing step to produce the whole output.

The Similarity Partitioning step is described in Algorithm 2. The operator initializes two lists of

Algorithm 2 Similarity Partitioning step for Blocking SJoin operator according to similarity function Sim_f and threshold γ

```
1:  procedure PARTITIONING
2:      Input: $\phi(D_A)$, $\mathrm{Sim}_f, \gamma$
3:      Output: List of $\mathrm{RSM}_A$, List of $\mathrm{RSM}_B$
4:      while getMolecule($\phi(D_A)$) do
5:          $\mathcal{M}_{iA} \leftarrow$ getMolecule($\phi(D_A)$)
6:          $R_{iA} \leftarrow$ head($\mathcal{M}_{iA}$)                          ▷ Get URI
7:          for $\mathrm{RSM}_{jB} \in$ List of $\mathrm{RSM}_B$ do
8:              $\mathrm{RSM}_{jB} = ((R_{jB}, T_{jB})[(R_{lA}, T_{lA})), \ldots, (R_{kA}, T_{kA})]$
9:              $R_{jB} \leftarrow$ head(head($\mathrm{RSM}_{jB}$))                ▷ Get URI
10:             if $\mathrm{Sim}_f(R_{jB}, R_{iA}) \geq \gamma$ then                ▷ Probe
11:                 tail($\mathrm{RSM}_{jB}$) $\leftarrow$ tail($\mathrm{RSM}_{jB}$) + ($\mathcal{M}_{iA}$)
        return sort(List of $\mathrm{RSM}_A$),sort(List of $\mathrm{RSM}_B$)
```

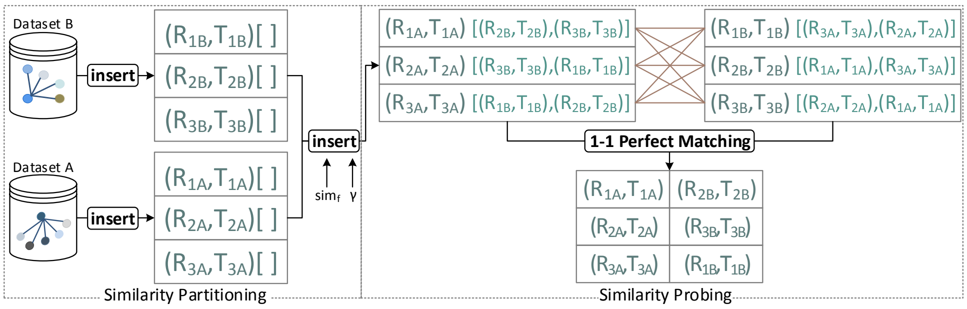


Figure 5.7: **SJoin Blocking Operator.** Similarity Partitioning step initializes lists of RSMs and populates their tails through a similarity function Sim_f and a threshold γ. Similarity Probing step performs 1-1 weighted perfect matching and outputs the perfect pairs of semantically equivalent molecules $(\mathcal{M}_{iA}, \mathcal{M}_{jB})$.

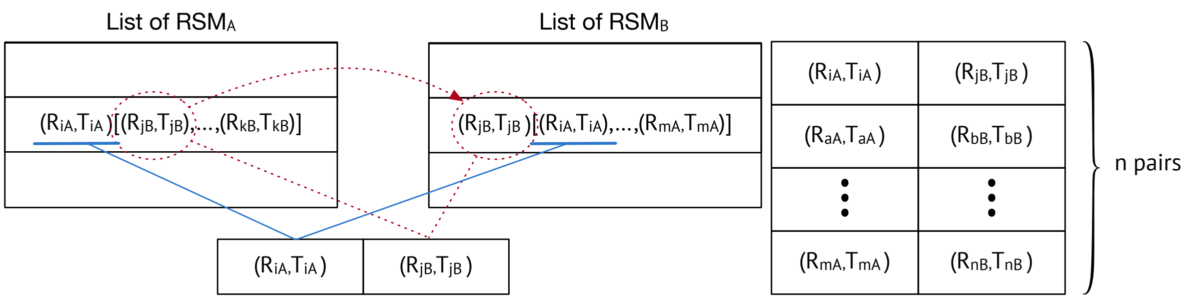
RSMs for two RDF graphs and incoming RDF molecules are inserted into a respective list with a filled head $\mathcal{M}$ and empty tail T. To populate the tail of a RSM in the list A, SJoin resorts to a semantic similarity function for computing a similarity score between the RSM and all RSMs in the opposite list B. If the similarity score exceeds a certain threshold γ then the molecule from the list B is appended to the tail of the RSM. Finally, the tail is sorted in the descending similarity score order such that the most similar RDF molecule obtains the top position in the tail. For instance, the semantic similarity function GADES [108] is able to decide relatedness between the RDF molecules of `dbr:Ibuprofen` and `drugbank:DB01050` in Fig. 5.6, and assigns a similarity score of 0.8. The algorithm supports datasets with arbitrary amounts of molecules. However, in order to guarantee 1-1 perfect matching, we place a restriction $card(\phi(D_A)) = card(\phi(D_B))$, i.e., the number of molecules in $\phi(D_A)$ and $\phi(D_B)$ must be the same. Thus, $card(\text{List of } RSM_A) = card(\text{List of } RSM_B)$.

A 1-1 weighted perfect matching is applied at the Similarity Probing stage in the Blocking SJoin operator. It accepts the lists of RSM_A, RSM_B created and populated during the previous Similarity Partitioning step. This step aims at producing perfect pairs of semantically equivalent RDF molecules $(\mathcal{M}_{iA}, \mathcal{M}_{jB})$, i.e., $max(Sim_f(\mathcal{M}_{iA}, RSM_B)) = max(Sim_f(\mathcal{M}_{jB}, RSM_A)) = Sim_f(\mathcal{M}_{iA}, \mathcal{M}_{jB})$. That is, for a given molecule $\mathcal{M}_{iA}$, there is no molecule in the list of RSM_A which has a similarity score higher than $Sim_f(\mathcal{M}_{iA}, \mathcal{M}_{jB})$ and vice versa. Algorithm 3 and Fig. 5.8 illustrate how perfect pairs are created.

Traversing the List of RSM_A, the algorithm iterates over each RSM_{iA}. Then, the tail of RSM_{iA}, i.e., an ordered list of *highly* similar molecules, is extracted. The first molecule of the tail RSM_{jB} corresponds to the most similar molecule from the List of RSM_B. The algorithm searches for RSM_{jB} in the List of RSM_B and examines whether the molecule (R_{iA}, T_{iA}) is the first one in the tail of RSM_{jB}. If this condition holds and (R_{iA}, T_{iA}) is not already matched with another RSM, then the pair $((R_{iA}, T_{iA}), (R_{jB}, T_{jB}))$ is identified as a perfect pair and is appended to the result list of pairs LP (cf. Fig. 5.8(a)). If false, then the algorithm finds the first occurrence of (R_{iA}, T_{iA}) in the tail of RSM_{jB} and appends the result pair to LP. When all $RSMs$ are matched, the algorithm yields the list of perfectly matched pairs (cf. Fig. 5.8(b)).

5.3.4 Non-Blocking SJoin Operator

The Non-Blocking SJoin operator aims at identifying $N - M$ matchings, i.e., an RSM_{iA} might be associated with multiple RSMs, e.g., RSM_{jB} or RSM_{kB}. Therefore, 1-1 weighted perfect matching is not executed which enables the operator to produce results as soon as new molecules arrive, i.e., in a non-blocking, on-demand manner. The operator receives two sets of RDF molecules $\phi(D_A)$ and $\phi(D_B)$. Lists of RSM_A, RSM_B are initialized as empty lists. Algorithm 4 describes the join procedure and Fig. 5.9 illustrates the algorithm.



(a) 1-1 matching from the bipartite graph of RMS

(b) Matched pairs

Figure 5.8: **1-1 Weighted Perfect Matching**. (a) The matching is identified from the lists of RSM_A and RSM_B; RDF molecules $\mathcal{M}_{iA}$=(R_{iA},T_{iA}) and $\mathcal{M}_{jB}$= (R_{jB},T_{jB}) are semantically equivalent whenever R_{iA} and R_{jB} are reciprocally the most similar RDF molecules according to Sim_f.

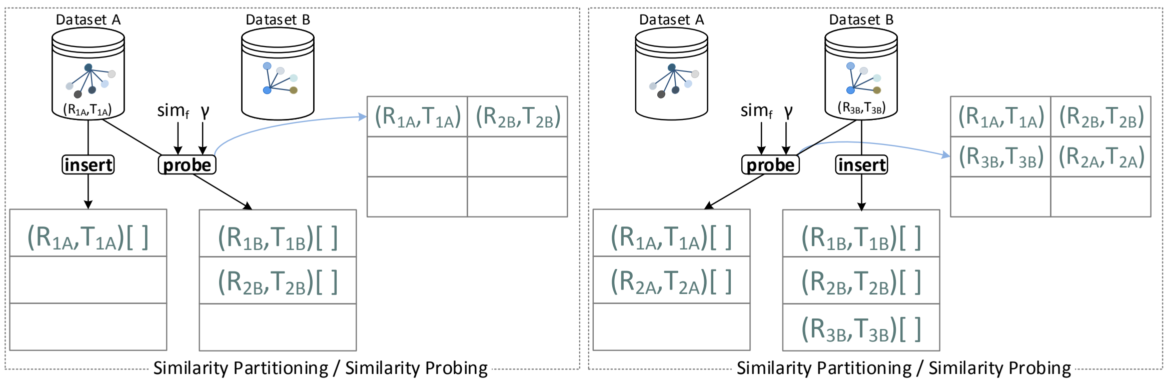
Algorithm 3 1-1 Weighted Perfect Matching of RSMs bipartite graph

```
 1: procedure MATCHING
 2:     Input: List of RSM$_A$, List of RSM$_B$
 3:     Output: List of pairs $LP = ((R_{iA}, T_{iA}), (R_{jB}, T_{jB}))$
 4:     for $RSM_{iA}$ ∈ List of RSM$_A$ do
 5:         RSM$_{iA}$ = $((R_{iA}, T_{iA})[(R_{jB}, T_{jB}), \ldots, (R_{kB}, T_{kB})])$        ▷ Ordered Set
 6:         for $(R_{jB}, T_{jB}) \in tail($RSM$_{iA})$ do
 7:             RSM$_{jB}$ ← Find in the List of RSM$_B$
 8:             RSM$_{jB}$ = $((R_{jB}, T_{jB})[(R_{lA}, T_{lA}), \ldots, (R_{zA}, T_{zA})])$    ▷ Ordered Set
 9:             if $(R_{lA}, T_{lA}) = (R_{iA}, T_{iA})$ and $(R_{iA}, T_{iA}) \notin LP$ then
10:                 $LP \leftarrow LP + ((R_{iA}, T_{iA}), (R_{jB}, T_{jB}))$            ▷ Add to result
11:             else
12:                 for $(R_{lA}, T_{lA}) \in tail($RSM$_{jB})$ do
13:                     find the position of $(R_{iA}, T_{iA})$
        return $LP$
```

For every incoming molecule $\mathcal{M}_{iA}$ from $\phi(D_A)$, Algorithm 4 performs the same two steps: *Similarity Partitioning* and *Similarity Probing*. The URI R_{iA} of an RDF molecule extracted from the tuple (R_{iA}, T_{iA}) is probed against URIs of *all* existing $RSMs$ in the List of RSM_B (cf. Fig. 5.9). If the similarity score of $Sim_f(R_{iA}, R_{jB})$ exceeds the threshold γ, then the pair $((R_{iA}, T_{iA}), (R_{jB}, T_{jB}))$ is considered as a matching and appended to the results list LP. During the *Similarity Insert* step, an RSM_{iA} is initialized, the molecule (R_{iA}, T_{iA}) becomes its head, and eventually added to the respective List of RSM_A. Algorithm 4 is applied to both $\phi(D_A)$ and $\phi(D_B)$ and able to produce results with constantly updating Lists of RSMs.

5.3.5 Time Complexity Analysis

The SJoin binary operator receives two RDF graphs of n RDF molecules each. To estimate the complexity of the blocking SJoin operator, three most expensive operations have to be analyzed. Table 5.4 gives an overview of the analysis. The complexity of the Data Partitioner module depends on the Algorithm 2, i.e., construction of Lists of RSM$_A$, RSM$_B$ and a similarity function Sim_f. The asymptotic approximation equals to $O(n^2 \cdot O(Sim_f))$. To produce ordered tails of RSMs the similar molecules in the tail have to be sorted in the descending similarity score order. The applicable merge sort and heapsort algorithms have



(a) Molecule (R_{iA}, T_{iA}) yields a pair $((R_{1A}, T_{1A}), (R_{2B}, T_{2B}))$

(b) Molecule (R_{3B}, T_{3B}) yields a pair $((R_{3B}, T_{3B}), (R_{2A}, T_{2A}))$

Figure 5.9: **SJoin Non-Blocking Operator.** Identifies N-M matchings and produces results as soon as new molecule arrives. When a molecule (R_{iA}, T_{iA}) arrives, it is inserted into a relevant list and probed against another list. If the similarity score exceeds the threshold γ, a new matching is produced.

Algorithm 4 The Non-Blocking SJoin operator executes both Similarity Partitioning and Probing steps as soon as an RDF molecule arrives from an RDF graph.

```
1:  procedure NON-BLOCKING OPERATOR
2:      Input: Dataset $\phi(D_A)$, Sim$_f, \gamma$
3:      Output: List of pairs $LP = ((R_{iA}, T_{iA}), (R_{jB}, T_{jB}))$
4:      while getMolecule($\phi(D_A)$) do
5:          $\mathcal{M}_{iA} \leftarrow$ getMolecule($\phi(D_A)$)
6:          $R_{iA} \leftarrow head(\mathcal{M}_{iA}), T_{iA} \leftarrow tail(\mathcal{M}_{iA})$          ▷ Get URI, tail
7:          for RSM$_{jB}$ ∈ List of RSM$_B$ do
8:              RSM$_{jB}$ = $((R_{jB}, T_{jB})[])$
9:              $R_{jB} \leftarrow head(head($RSM$_{jB}))$          ▷ Get URI
10:             $T_{jB} \leftarrow tail(head($RSM$_{jB})$          ▷ Get tail
11:             if Sim$_f(R_{iA}, R_{jB}) \geq \gamma$ then          ▷ Probe
12:                 $LP \leftarrow LP + ((R_{iA}, T_{iA}), (R_{jB}, T_{jB}))$
13:         $head($RSM$_{iA}) \leftarrow \mathcal{M}_{iA}$, $tail($RSM$_{iA}) \leftarrow []$
14:         List of RSM$_A$ ←List of RSM$_A$+RSM$_{iA}$          ▷ Insert
        return $LP$
```

$O(n \log n)$ asymptotic complexity. The 1-1 Weighted Perfect Matching component has $O(n^3)$ complexity in the worst case according to the Algorithm 3. However, the Hungarian algorithm [167], a standard approach for 1-1 weighted perfect matching, converges to the same $O(n^3)$ complexity. Partitioning, sorting, and perfect matching are executed sequentially. Therefore, the overall complexity conforms to the sum of complexities, i.e., $O(n^2 \cdot O(Sim_f)) + O(n \log n) + O(n^3)$ which equals to $O(n^2 \cdot O(Sim_f)) + O(n^3)$. We thus deduce that the *SJoin* complexity depends on the complexity of a chosen similarity measure whereas the lowest achievable order of complexity is limited to $O(n^3)$.

The complexity of the non-blocking SJoin operator stems from the analysis of the Algorithm 4. The most expensive step of the algorithm is to compute a similarity score between an RSM$_{iA}$ and RSMs in the List of RSM$_B$. Applied to both $\phi(D_A)$ and $\phi(D_B)$ the complexity converges to $O(n^2 \cdot O(Sim_f))$.

Table 5.4: The *SJoin* Time Complexity. Results for the steps of Partitioning, Sorting, and Matching, where n is the number of RDF molecules.

Stage	**Blocking SJoin Complexity**	**Non-Blocking SJoin Complexity**
Partitioning	$O(n^2 \cdot O(Sim_f))$	$O(n^2 \cdot O(Sim_f))$
Sorting	$O(n \log n)$	
Matching	$O(n^3)$	
Overall	$O(n^2 \cdot O(Sim_f)) + O(n^3)$	$O(n^2 \cdot O(Sim_f))$

Sections 5.1, 5.2, 5.3 address the steps of the knowledge integration and fusion pipeline in Fig. 5.2. Having performed knowledge extraction, integration, and interlinking of heterogeneous graphs obtained from numerous sources, a resulting knowledge graph benefits from increased consistency and tends to provide a holistic view on the sources. In the following Section, we provide experimental results of the knowledge extraction methodology from Web tables and SJoin operator. In the following Chapter 6 we tackle problems of effective and efficient query processing over knowledge graphs.

5.4 Experimental Study

5.4.1 Evaluating Knowledge Extraction from Tables

The main goal of the evaluation is to assess the performance of the proposed methodology in comparison with the existing solutions. By the end of the section we decide whether the hypothesis made in Section 5.1.1 is demonstrated or not. The evaluation consists of two subgoals – the evaluation of machine learning algorithms with string similarity functions and the evaluation of the system as a whole.

Benchmark: Four machine learning algorithms are compared: *Naive Bayes* classifier is a simple and popular machine learning algorithm. It is based on Bayes theorem with naive assumptions regarding independence between parameters presented in a training set. However, this ideal configuration rarely occurs in real datasets so that the result always has a statistical error [164]. *A decision tree* is a predictive model that is based on tree–like graphs or binary trees [168]. Branches represent a conjunction of features and a leaf represents a particular class. Going down the tree we eventually end up with a leaf (a class) with its own unique configuration of the features and values. *k-nearest neighbours* is a simple classifier based on a distance between objects [169]. If an object might be represented in Euclidean space then there is a number of functions that could measure a distance between these objects. If the majority of neighbours of the object belongs to one class than the object would be classified into the same class. *Support Vector Machine* is a non–probabilistic binary linear classifier that tries to divide instances of classes presented in a training set by a gap as wide as possible. In other words, SVM builds separating surfaces between categories, which might be linear or non-linear [164].

For each of the machine learning algorithms we apply three different previously described string similarity techniques, i.e., *Levenshtein*, *Jaro-Winkler* distances, and *n-grams*.

Dataset: A training set of 400 tables taken from the corpus[7] as a result of [161] was prepared to test suggested heuristics and machine learning methods.

Metrics: We measured *Precision*, *Recall* and *F–Measure* in order to evaluate the performance of the genuineness check and orientation identification.

Implementation: The proposed approach was implemented in Java as a plugin for the Information

[7]WDC – Web Tables. Web: http://webdatacommons.org/webtables/

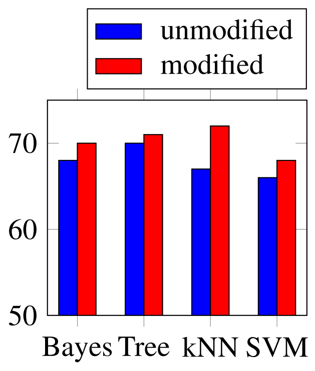
Table 5.5: Evaluation of the genuineness check

Method			**Precision**	**Recall**	**F-Measure**
Naive Bayes	Levenshtein	unmodified	0.925	0.62	0.745
		modified	0.93	0.64	0.76
	Jaro-Winkler	unmodified	0.939	0.613	0.742
		modified	0.939	0.617	0.744
	n-grams	unmodified	0.931	0.633	0.754
		modified	0.937	0.643	0.763
Decision Tree	Levenshtein	unmodified	0.928	0.65	0.765
		modified	0.942	0.653	0.76
	Jaro-Winkler	unmodified	0.945	0.637	0.761
		modified	0.946	0.64	0.763
	n-grams	unmodified	0.933	0.603	0.733
		modified	0.945	0.637	0.76
kNN	Levenshtein	unmodified	0.904	0.623	0.74
		modified	0.943	0.667	0.78
	Jaro-Winkler	unmodified	0.928	0.607	0.734
		modified	0.941	0.64	0.762
	n-grams	unmodified	0.948	0.663	0.78
		modified	0.949	0.677	0.79
SVM	Levenshtein	unmodified	0.922	0.597	0.725
		modified	0.93	0.62	0.744
	Jaro-Winkler	unmodified	0.924	0.61	0.735
		modified	0.926	0.623	0.745
	n-grams	unmodified	0.922	0.627	0.746
		modified	0.927	0.637	0.755

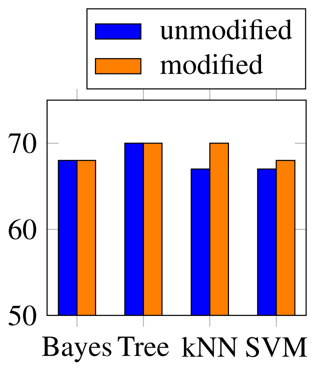
Workbench[8] platform developed by fluidOps. The platform provides numerous helpful APIs responsible for the user interaction, RDF data maintenance, ontology generation, knowledge bases adapters and smart data analysis. The machine learning algorithms are supplied by *WEKA*[9] – a comprehensive data mining Java framework developed by the University of Waikato. The *SimMetrics*[10] library by UK Sheffield University provided string similarity functions.

Table Classification Performance. Table 5.5 and Fig. 5.10, Fig. 5.11 show the obtained experimental results. Fig. 5.10 represents the overall fracture of correctly classified genuine and non-genuine tables w.r.t. used machine learning algorithms and string similarity functions. The machine learning algorithm based on kNN in conjunction with Levenshtein distance or n-grams demonstrated the highest efficiency during the genuineness check. A slight increase in efficiency in spite of modifications is observed mostly for kNN. It also could be noted that overall results of classification are generally lower in comparison with orientation classification task. This may indicate a lack of information about the table structure caused by a small amount of heuristics. Development and implementation of more sophisticated numerical parameters is suggested in order to improve the performance of classification. Hence, the way towards

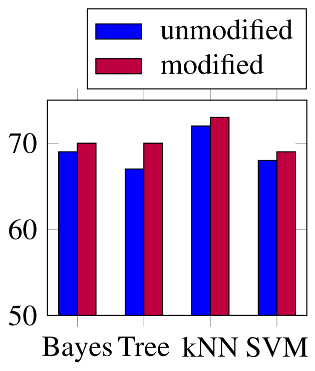
[8]Web: `http://www.fluidops.com/en/portfolio/information_workbench/`
[9]Web: `http://www.cs.waikato.ac.nz/ml/weka/`
[10]Web: `http://sourceforge.net/projects/simmetrics/`



(a) Levenshtein

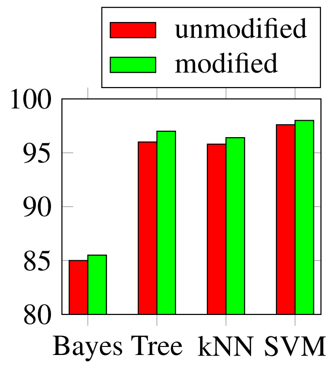


(b) Jaro-Winkler

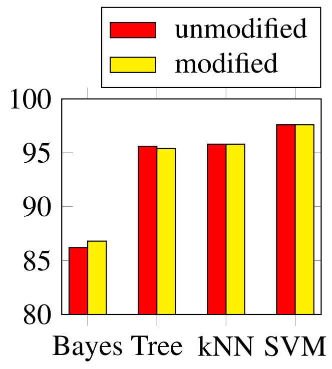


(c) n-grams

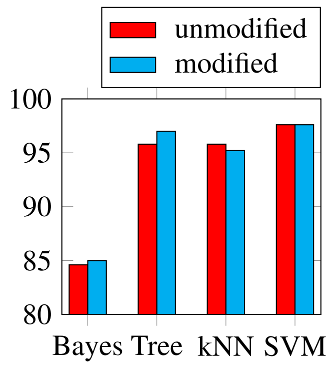
Figure 5.10: **Genuineness check, correctly classified, %**



(a) Levenshtein



(b) Jaro-Winkler



(c) n-grams

Figure 5.11: **Orientation check, correctly classified, %**

improving overall F–Measure is connected with raising Recall of the approach.

Fig. 5.11 indicates the high efficiency of the orientation check task. Most of the used machine learning methods except Naive Bayes demonstrated close to 100% results. A relatively low result of Naive Bayes regardless of the chosen string similarity function might be explained by a number of assumptions which the method is established on. important details which affect the classification process because of such simplicity. During the orientation check only genuine tables are considered and assessed. Therefore, the eventual result is Precision.

Having analyzed the efficiency of machine learning methods with string metric mechanisms we decided to apply modified kNN in conjunction with Levenshtein distance during the genuineness check process and modified SVM in conjunction with Levenshtein distance during the orientation check process.

The overall performance of the approach is defined as a product of the highest *F–Measure* of the genuineness check and the highest *Precision* of the orientation check, which results in *0.77* or *77%*. It indicates, that we are able to correctly extract knowledge at least from three of given four arbitrary tables.

Taking into account the achieved results we consider the hypothesis suggested in Section 5.1.1 demonstrated. Indeed, unstructured data contains semantics. Hence, the following questions are raised: *How much semantics does unstructured data contain? Is there an opportunity to semantically integrate tables with other types of Web content?* Answering the questions will facilitate the shift from neglecting the tables towards close integration of all the Web content.

Table 5.6: Benchmark Description. RDF datasets used in the evaluation.

	Experiment 1: People		Experiment 2: People			
	DBpedia D1	DBpedia D2	DBpedia	Wikidata	DBpedia	Wikidata
Molecules	500	500	500	500	1000	1000
Triples	17,951	17,894	29,263	16,307	54,590	29,138

5.4.2 Evaluating SJoin

An empirical evaluation is conducted to study the efficiency and effectiveness of SJoin in blocking and non-blocking conditions on DBpedia and Wikidata. We assess the following questions: **Q**1) Does blocking SJoin integrate RDF graphs more efficiently and effectively compared to the state of the art? **Q**2) What is the impact of threshold values on the completeness of a non-blocking SJoin? **Q**3) What is the effect of a similarity function in the SJoin results? The experimental configuration:

Benchmark: Experiment 1 is executed against a dataset of 500 molecules[11] of type Person extracted from the live version of DBpedia (February 2017). Based on the original molecules, we created two sets of molecules by randomly deleting or editing triples in the two sets. Sharing the same DBpedia vocabulary, Experiment 1 datasets have a higher resemblance degree compared to Experiment 2. Experiment 2 employs subsets of DBpedia and Wikidata of the Person class. Assessing SJoin in the higher heterogeneity settings, we sampled datasets of 500 and 1000 molecules varying triples count from 16K up to 55K[12]. Table 5.6 provides statistics on the datasets. DBpedia D1 and D2 refer to the dumps of 500 molecules. The dumps of 500 and 1000 molecules for Experiment 2 are extracted from DBpedia and Wikidata.

Baseline: Gold standards for blocking operators comparison include the original DBpedia Person descriptions (Experiment 1) and `owl:sameAs` links between DBpedia and Wikidata (Experiment 2). We compare SJoin with a Hash Join operator. For a fair comparison, the Hash Join was extended to support similarity functions at the *Probing* stage. That is, blocking SJoin is compared against blocking similarity Hash Join and non-blocking SJoin is evaluated against non-blocking Symmetric Hash Join. The Gold standard for evaluating non-blocking operators is comprised of the precomputed amounts of pairs which similarity score exceeds a predefined threshold; gold standards are computed off line.

Metrics: We report on execution time (ET in secs) as the elapsed time required by the SJoin operator to produce all the answers. Furthermore, we measure *Precision*, *Recall* and report *F1-measure* during the experiments with blocking operators. Precision is the fraction of RDF molecules that has been identified and integrated (M) that intersects with the Gold Standard (GS), i.e., $Precision = \frac{|M \cap GS|}{|M|}$. Recall corresponds to the fraction of the identified similar molecules in the Gold Standard, i.e., $Recall = \frac{|M \cap GS|}{|GS|}$. Comparing non-blocking operators, we measure *Completeness* over time, i.e., a fraction of results produced at a certain time stamp. The timeout is set to one hour (3,600 seconds), the operators results are checked every second. Ten thresholds in the range [0.1 : 1.0] and step 0.1 were applied in Experiment 1. In Experiment 2, five thresholds in the range [0.1 : 0.5] were evaluated because no pair of entities in the sampled RDF datasets has a GADES similarity score higher than 0.5.

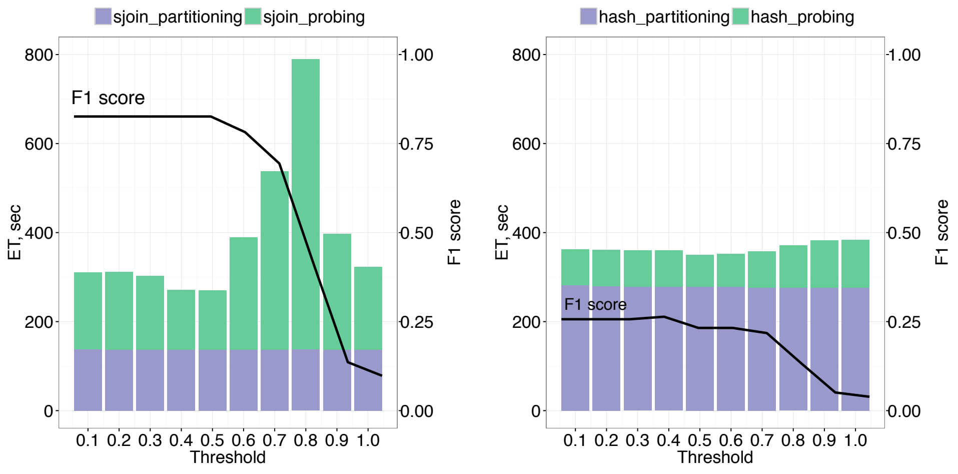
Implementation: Both blocking and non-blocking SJoin operators are implemented in Python 2.7.10[13]. Baseline improved Hash Joins are implemented in Python as well[14]. The experiments were executed on a Ubuntu 16.04 (64 bits) Dell PowerEdge R805 server, AMD Opteron 2.4GHz CPU, 64

[11] `https://github.com/RDF-Molecules/Test-DataSets/tree/master/DBpedia-People/20160819`

[12] `https://github.com/RDF-Molecules/Test-DataSets/tree/master/DBpedia-WikiData/operators_evaluation`

[13] `https://github.com/RDF-Molecules/operators/tree/master/mFuhsion`

[14] `https://github.com/RDF-Molecules/operators/tree/master/baseline_ops`



(a) SJoin performance (b) Hash Join performance

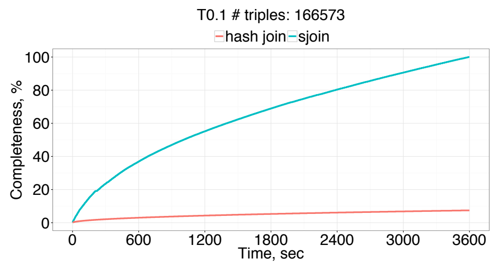
Figure 5.12: **Experiment 1 (GADES) with blocking operators**. The *partitioning* bar shows the time taken to partition the molecules in RSMs, *probing* indicates the time required for 1-1 weighted perfect matching. Black line chart on the right axis denotes F1 score. (a) SJoin demonstrates higher F1 score while consuming more time for perfect matching. (b) Baseline Hash Join demonstrates less than 0.25 F1 score even on lower thresholds spending less time on probing.

cores, 256GB RAM. We evaluated two similarity functions: GADES [108] and Semantic Jaccard (SemJaccard) [170]. GADES relies on semantic descriptions encoded in ontologies to determine relatedness, while SemJaccard requires the materialization of implicit knowledge and mappings. Evaluating schema heterogeneity of DBpedia and Wikidata in Experiment 2 the similarity function is fixed to GADES.

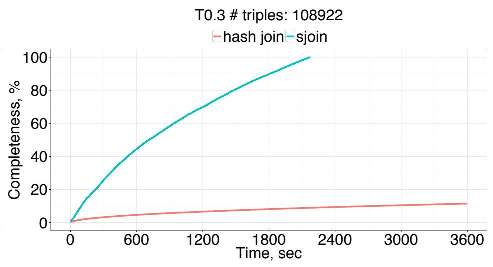
5.4.3 Experiment 1 - DBpedia – DBpedia People

Experiment 1 evaluates the performance and effectiveness of blocking and non-blocking SJoin compared to respective Hash Join implementations. The testbed includes two split DBpedia dumps with semantically equivalent entities but non-matching resource URIs and randomly distributed properties; GADES and SemJaccard similarity functions. That is, both graphs are described in terms of one DBpedia ontology. Fig. 5.12 visualizes the results obtained when applying GADES semantic similarity function in order to identify a perfect matching of graphs resources, i.e., in blocking conditions. SJoin exhibits better F1 score up to very high 0.9 threshold value. Moreover, the effectiveness of more than 80% is ensured up to 0.6 threshold value whereas Hash Join barely reaches 25% even on lower thresholds. The partitioning time is constant for both operators but Hash Join performs the partitioning slower due to the application of a hash function to all incoming molecules. However, high effectiveness of SJoin is achieved at the expense of time efficiency. SJoin has to complete a 1-1 perfect matching algorithm against a large 500x500 matrix whereas Hash Join performs the perfect matching *three* times but for smaller matrices equal to the size of its buckets, e.g., about 166x166 for three buckets which is faster due to the cubic complexity of the weighted perfect matching algorithm.

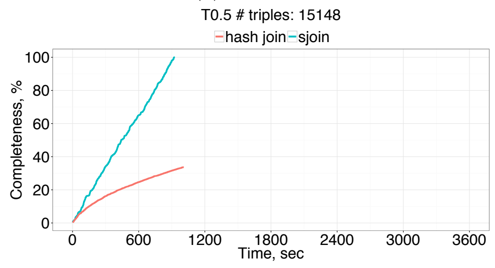
Fig. 5.13 shows the results of the evaluation of non-blocking operators with GADES. SJoin outperforms the baseline Hash Join in terms of completeness over time in all four cases with the threshold in the range 0.1-0.8. Fig. 5.13(a) demonstrates that the SJoin operator is capable of producing 100% of results within the timeframe whereas the Hash Join operator outputs only about 10% of the expected tuples. In



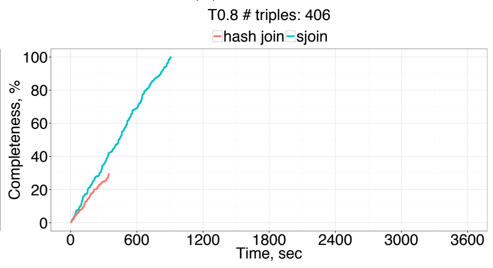
(a) T = 0.1



(b) T = 0.3



(c) T = 0.5



(d) T = 0.8

Figure 5.13: **Experiment 1 (GADES) with non-blocking operators.** SJoin produces complete results at all threholds in contrast to Hash Join.

Fig. 5.13(b), SJoin achieves the full completeness even faster. In Fig. 5.13(c) both operators finish after 18 minutes, but SJoin retains full completeness while Hash Join reaches only 35%. Finally, with the 0.8 threshold in Fig.5.13(d), Hash Join performs very fast but still struggles to attain the full completeness; SJoin takes more time but sustainably achieves answer completeness. One of the reasons why Hash Join performs worse is its hash function which does not consider semantics encoded in the molecules descriptions. Therefore, the hash function partitions RDF molecules into buckets almost randomly, while it was originally envisioned to place similar entities in the same buckets.

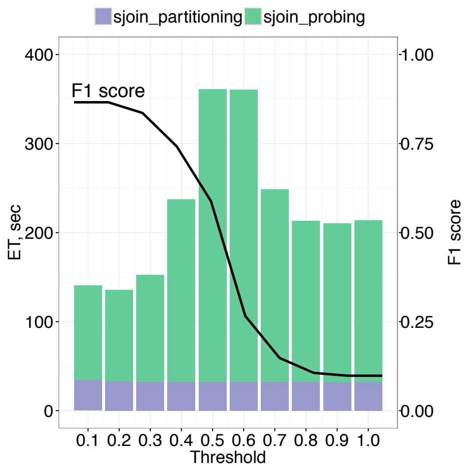
Fig. 5.14 presents the efficiency and effectiveness of blocking SJoin and Hash Join when applying Sem-Jaccard similarity function. As an unsophisticated measure, operators require less time for partitioning and take less time for probing stages. That is, due to the heterogeneous nature of the compared datasets, SemJaccard is not able to produce similarity scores higher than 0.4. On the other hand, SemJaccard simplicity leads to significant deterioration of the F1 score already at low thresholds, i.e., 0.3-0.4.

Fig. 5.15 illustrates the difference in elapsed time and achieved completeness of SJoin and Hash Join applying GADES or SemJaccard similarity functions. Evidently, SemJaccard outputs fewer tuples even on lower thresholds, e.g., 486 pairs at 0.4 threshold against 50,857 pairs by GADES. We therefore demonstrate that plain set similarity measures as SemJaccard that consider only an intersection of exactly same triples are ineffective in integrating heterogeneous RDF graphs.

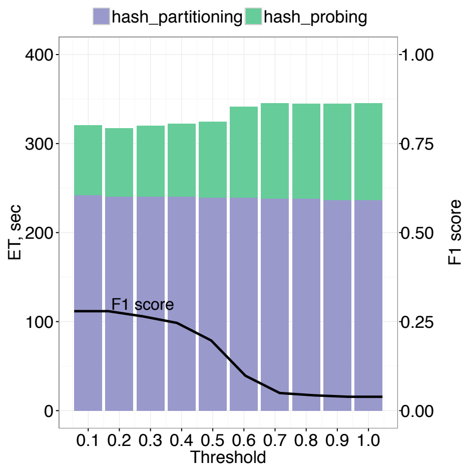
5.4.4 Experiment 2 - DBpedia - Wikidata People

The distinctive feature of the experiment consists in completely different vocabularies used to semantically describe the same people. Therefore, traditional joins and set similarity joins, e.g., Jaccard, are not applicable. We evaluate the performance of SJoin employing GADES semantic similarity measure.

Fig. 5.16 reports the efficiency and effectiveness of SJoin compared to Hash Join in the 500 molecules

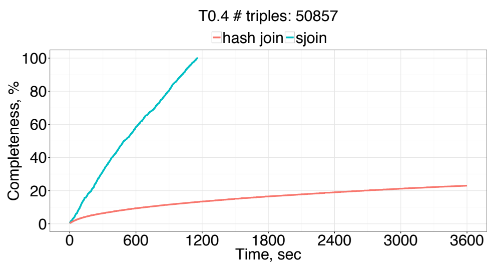


(a) SJoin performance

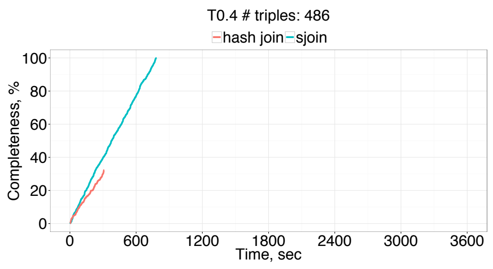


(b) Hash Join performance

Figure 5.14: **Experiment 1 (SemJaccard) with blocking operators**. (a) SJoin takes less time to compute similarity scores while F1 score quickly deteriorates after threshold 0.5. (b) Baseline Hash Join in most cases consumes more time and produces less reliable matchings.



(a) T = 0.4, GADES



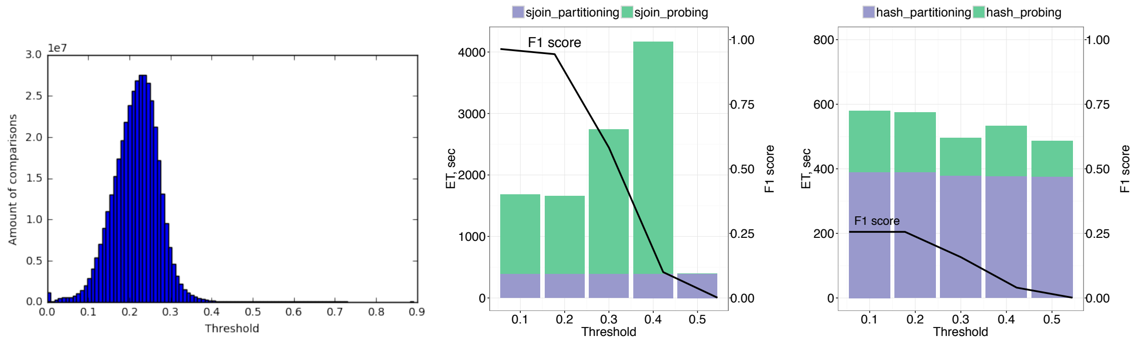
(b) T = 0.4, Jaccard

Figure 5.15: **Experiment 1 with fixed threshold**. GADES identifies two orders of magnitude more results than Jaccard while SJoin still achieves full completeness.

setup. Fig. 5.16(a) justifies the range of selected thresholds as only a few number of pairs have a similarity score higher than 0.5. Blocking SJoin manages to achieve higher F1 score (max 95%) up to 0.3 threshold value, but requires significantly more time to accomplish the perfect matching.

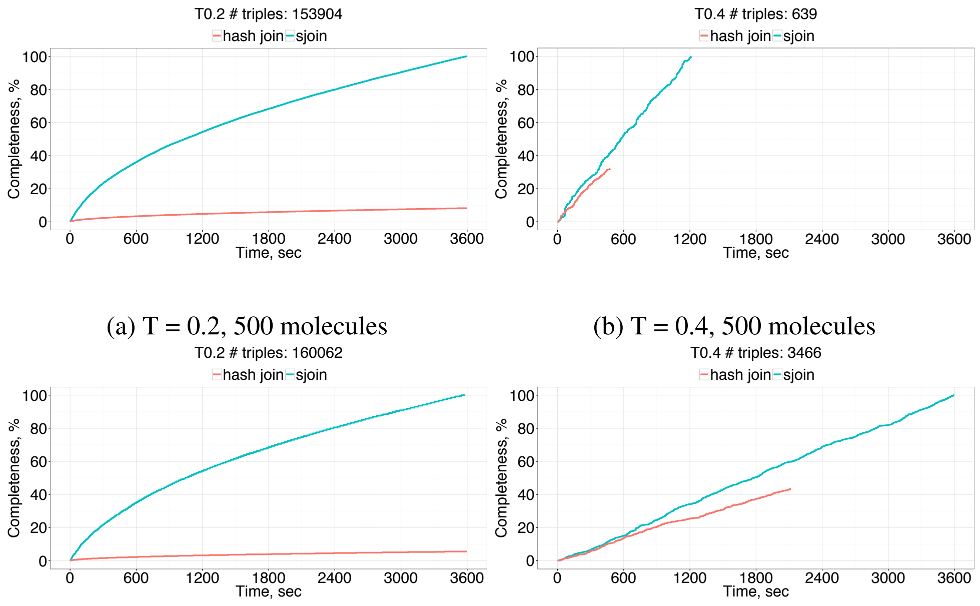
Results of non-blocking SJoin and Hash Join executed against 500 and 1000 molecules configurations are reported in Fig. 5.17. The observed behavior of these operators resembles the one in Experiment 1, i.e., SJoin outputs complete results within a predefined time frame, while Hash Join barely achieves 40% completeness in the case with a relatively high threshold 0.4 and small number of outputs.

Analyzing the observed empirical results, we are able to answer our research questions: **Q**1) Blocking SJoin consistently exhibits higher F1 scores, and the results are more reliable. However, time efficiency depends on the input graphs and applied similarity functions. **Q**2) A threshold value prunes the amount of expected results and does not affect the completeness of SJoin. **Q**3) Clearly, a semantic similarity function allows for matching RDF graphs more accurately.



(a) GADES distribution (b) SJoin (c) Hash Join

Figure 5.16: **Experiment 2 (GADES) with blocking operators, 500 molecules**. (a) The distribution of GADES similarity scores shows that there are few pairs which score exceeds 0.4 threshold. (b) SJoin requires more time but achieves more than 0.9 F1 score until T0.3. (c) Baseline Hash Join is faster but its F1 score is below 0.25.



(a) T = 0.2, 500 molecules (b) T = 0.4, 500 molecules

(c) T=0.2, 1000 molecules (d) T=0.4, 1000 molecules

Figure 5.17: **Experiment 2. Non-blocking operators in different dataset sizes.** In larger setups, SJoin still reaches full completeness.

5.5 Summary

In this Chapter, we addressed challenges arising when building knowledge graphs. A typical pipeline includes knowledge extraction, integration, and interlinking steps. We emphasized importance and applicability of the pipeline for integrating heterogeneous RDF graphs.

The suggested approach knowledge extraction from tables is highly effective in table structure analysis tasks and transformation of raw tabular data into a valid RDF graph. To sum up, the methodology enjoys the following benefits: 1) Automatic extraction of HTML tables from the sources specified by a user; 2) Implementation of string metrics and machine learning algorithms to analyze genuineness and structure of a table; 3) Automatic RDF graph generation and publishing of the extracted dataset.

Once knowledge is extracted, resulting graphs have to be integrated and interlinked. The described semantic data integration framework caters for those two steps providing an architecture and a set of concepts, such as RDF molecules, fusion policies, semantic similarity functions, which we adopt for the physical semantic join operator. Further, we presented SJoin, an operator for detecting semantically equivalent RDF molecules from RDF graphs. SJoin implements two operators: Blocking and Non-Blocking, which rely on similarity measures and ontologies to effectively detect equivalent entities from heterogeneous RDF graphs. Moreover, the time complexity of SJoin operators depends on the time complexity of the similarity measure, i.e., SJoin does not introduce additional overhead. The behavior of SJoin was empirically studied on DBpedia and Wikidata real-world RDF graphs, and on Jaccard and GADES similarity measures. Observed results suggest that SJoin is able to identify and merge semantically equivalent entities, and is empowered by the semantics encoded in ontologies and exploited by similarity measures. Therefore, the resulting computational framework can be applied for semantic integration of heterogeneous RDF graphs.

The approaches presented in the Chapter empirically demonstrate that semantics encoded in enterprise data indeed facilitates its integration into RDF graphs. Therefore, we are able to answer the **RQ2**, i.e., semantic data integration techniques can be effectively applied with Enterprise Knowledge Graphs in order to lift and integrate non-semantic data. In the following Chapter 6, we tackle challenges of efficient and effective query processing as querying remains one of the main communication interfaces between a knowledge graph and external services.

CHAPTER 6

Query Processing over Knowledge Graphs

Previous Chapters, i.e., Chapter 4 and Chapter 5, focused on building and integrating knowledge graphs. Considering that graphs already contain and encode actionable knowledge, such a knowledge has to be properly leveraged in order to be *actionable*, that is, provide valuable insights to support the decision-making process. Querying remains the main communication mechanism between an EKG and external applications. Whereas logically an EKG is a unified *black box* that encapsulates the knowledge, physically large graphs are often scattered among numerous machines in a distributed or federated manner. This Chapter studies the Query Processing level of the architecture (cf. Fig. 6.1). As SPARQL is a standard language for querying RDF data, we investigate how SPARQL queries performance against large knowledge graphs can be improved taking into account characteristics of knowledge graphs and existing integration basis. At a higher level, the following research question is addressed in this Chapter:

Research Question 3 (RQ3)

How can we perform semantic query processing of federated enterprise data efficiently?

To address scalability, in this Chapter we focus on federated environments. In federated query processing a query engine has to perform four main stages: 1) *Query Decomposition* aims at breaking down an initial query in sub-groups that can often be executed in parallel according to a certain configuration. 2) Source Selection relies on this configuration to select relevant sources of the federated environment that will be able to answer a sub-query (made of sub-groups) with maximum achievable completeness. 3) During *Query Planning*, the engine tries to arrange a sequence of source calls and intermediate results

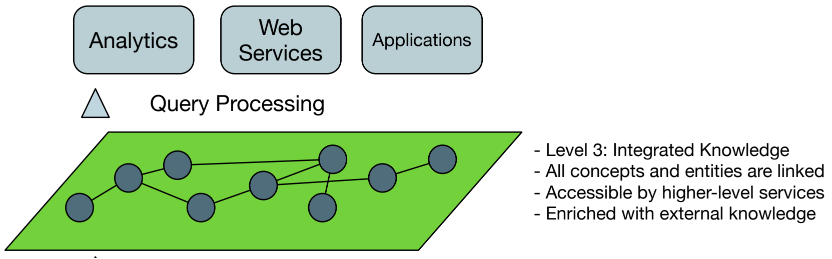

Analytics
Web Services
Applications
Query Processing
- Level 3: Integrated Knowledge
- All concepts and entities are linked
- Accessible by higher-level services
- Enriched with external knowledge

Figure 6.1: **The levels of the research problem addressed in this Chapter.** We concentrate on knowledge integration and fusion tasks in order to achieve a unified view on the sources of a knowledge graph (i.e., Level 3).

processing in a way that increases efficiency and effectiveness. 4) Finally, in *Query Execution* the plan built at the previous step is actually executed and processed for obtaining real answers from a graph.

As we showed in Chapter 3, federated SPARQL engines do not fully leverage semantics of RDF data. Similarly, query processing techniques well known in the relational database community and proved to be efficient are yet to appear in SPARQL engines. In this Chapter, we propose several novel techniques to improve federated query processing in SPARQL, i.e., fine-grained query optimization approaches that include a multi-way join operator and query planner, and an operator that combines both semantic similarity matching and arbitrary input arity. The proposed improvements adopt the concept of RDF molecules (defined in Chapter 5) and leverage an architecture of a federated query engine presented in [171] that caters for source selection, query decomposition and query execution stages.

Towards addressing the **RQ3** the following contributions are made:

- *SMJoin*, the first general-purpose multi-way join operator for SPARQL queries;
- A new SPARQL query planning technique that employs both binary and multi-way join operators within one query tree plan;
- *MSimJoin*, the first SPARQL operator that combines similarity-based fuzzy matching with an arbitrary number of inputs.

The claims made in this Chapter have been reviewed by the community and published in the following articles [171–173] [1].

The Chapter is structured as follows. Section 6.1 presents a background framework our contributions based on, i.e., a federated SPARQL query engine that employs a concept of *RDF molecule templates* in order to accelerate query decomposition and source selection steps. The framework can be extended and improved at every stage of the query processing workflow. We then look in details at the query planning stage proposing a multi-way join operator unknown in SPARQL query processing before in Section 6.2,. Additionally, we study the best application cases for binary and multi-way join operators. Furthermore, we investigate an opportunity to combine both binary and n-ary joins within one query plan that was not used in SPARQL before. Section 6.3 aims at merging benefits of the semantic similarity join operator presented in Chapter 5 and a multi-way join operator, i.e., take advantage of both similarity joins and multi-way joins in one physical operator. We report empirical evaluation results on the proposed optimizations implemented within a background federated query engine in Section 6.4 and provide a summary of the Chapter in Section 6.5.

[1][172] is a joint work with Kemele M. Endris, a PhD student from Leibniz University in Hannover, and Diego Collarana, a PhD student at the University of Bonn. In this paper, I contributed to the definition, description, and implementation of the multi-way join operator, problem formalization, preparation of datasets for experiments, analysis of obtained results, and reviewing related work.
[171] is a joint work with Kemele M. Endris, a PhD student from Leibniz University in Hannover, and Mohamed Nadjib Mami, a PhD student at the University of Bonn. In this paper, I contributed to the motivating example, definition of algorithms, visual description of the proposed approach, preparation of infrastructure and datasets for experiments, and reviewing related work.

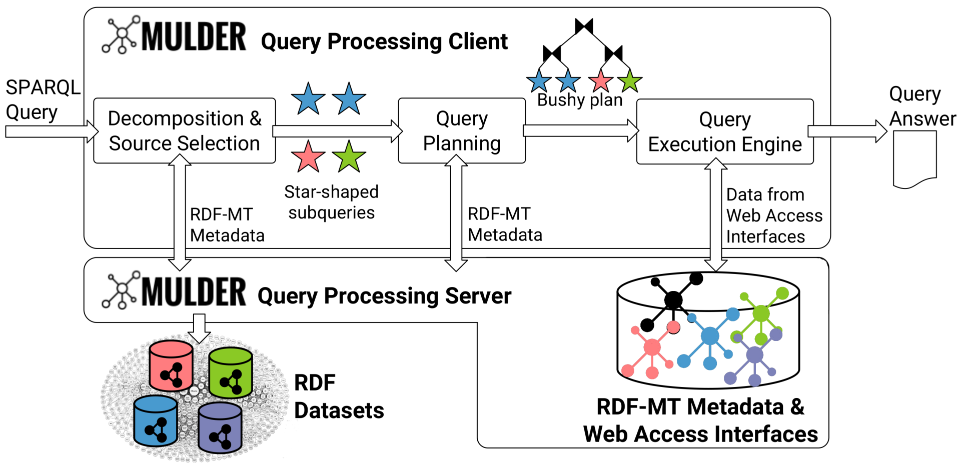

MULDER Query Processing Client
SPARQL Query
Decomposition & Source Selection
Star-shaped subqueries
Query Planning
Bushy plan
Query Execution Engine
Query Answer
RDF-MT Metadata
RDF-MT Metadata
Data from Web Access Interfaces
MULDER Query Processing Server
RDF Datasets
RDF-MT Metadata & Web Access Interfaces

Figure 6.2: **The Client-Server Architecture of the Engine**. The query processing client receives SPARQL queries, creates query decompositions with star-shaped subqueries, and identifies and executes bushy plans. The query processing server collects both RDF-MT metadata about RDF datasets and results of executing queries over Web access interfaces, e.g., SPARQL endpoints.

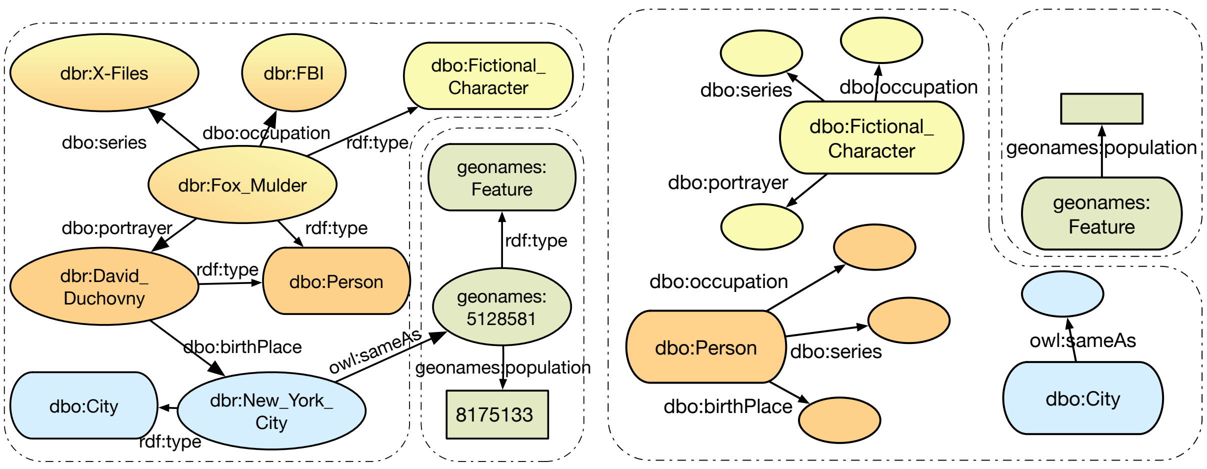
6.1 Background: A Federated SPARQL Query Engine

In this Section[2], we present a federated SPARQL query engine as a necessary background that is leveraged in the following Sections 6.2, 6.3. From [171] we adopt a notion of *RDF Molecule Templates* that support sources description, selection, and query decomposition. The experiments conducted in Section 6.4 employ this query engine for the described below query planning and optimization techniques.

Federated query processing has already been investigated by the database community and is currently receiving wide attention within semantic technology research [95, 96, 112, 174]. There are two major challenges in federated query processing: (i) Discovering the structure of the available RDF datasets for facilitating the source selection process; (ii) Optimizing query decomposition and planning for balancing between query answer completeness and query execution time. We describe an integrated approach for federated query processing (implemented in the MULDER system), which uses *RDF Molecule Templates* (RDF-MTs) for describing the structure of RDF datasets, and for bridging between parts of a query to be executed in a federated manner.

The architecture of the query engine is depicted in Fig. 6.2. The *Query Processing Client* receives a SPARQL query, and identifies and executes a bushy plan against RDF datasets. The *Decomposition & Source Selection* creates a query decomposition with service graph patterns (SGPs) of star-shaped subqueries built according to RDF-MT metadata. RDF-MTs describe the properties of the RDF molecules contained in the RDF datasets, where an RDF molecule is a set of RDF triples that share the same subject. Once the star-shaped subqueries are identified, a bushy plan is built by the *Query Planning*; the plan leaves correspond to star-shaped subqueries. The *Query Engine* executes the bushy plan and contacts the query processing server to evaluate SGPs over the Web access interfaces. Further, the server receives requests from the client to retrieve RDF-MT metadata about RDF datasets, e.g., metadata about properties of RDF molecules contained in these RDF datasets.

[2]This Section is based on [171], a joint work with Kemele M. Endris, a PhD student from Leibniz University in Hannover, and Mohamed Nadjib Mami, a PhD student at the University of Bonn. In this paper, I contributed to the motivating example, definition of algorithms, visual description of the proposed approach, preparation of infrastructure and datasets for experiments, and reviewing related work.



(a) Two RDF graphs (b) RDF Molecule Templates (RDF-MTs)

Figure 6.3: **RDF-MT Creation**. Two RDF graphs with four RDF molecules of types: `dbo:Person`, `dbo:City`, `geonames:Feature`, and `dbo:FictionalCharacter`. Four RDF Molecule Templates (RDF-MTs) are created for these RDF classes.

RDF Molecule Template (RDF-MT)

Definition 6.1.1 *An RDF Molecule Template (RDF-MT) is a 5-tuple=<WebI,C,DTP,IntraL,InterL>, where:*

- *WebI – is a Web service API that provides access to an RDF dataset G via SPARQL protocol;*
- *C – is an RDF class such that the triple pattern (?s rdf:type C) is true in G;*
- *DTP – is a set of pairs (p, T) such that p is a property with domain C and range T, and the triple patterns (?s p ?o) and (?o rdf:type T) and (?s rdf:type C) are true in G;*
- *IntraL – is a set of pairs (p,C_j) such that p is an object property with domain C and range C_j, and the triple patterns (?s p ?o) and (?o rdf:type C_j) and (?s rdf:type C) are true in G;*
- *InterL – is a set of triples (p,C_k,SW) such that p is an object property with domain C and range C_k; SW is a Web service API that provides access to an RDF dataset K, and the triple patterns (?s p ?o) and (?s rdf:type C) are true in G, and the triple pattern (?o rdf:type C_k) is true in K.*

Fig. 6.3 illustrates the creation of four RDF-MTs from two given RDF graphs (Fig. 6.3(a)). As technical details of molecule creation are out of the scope of this Section, Fig. 6.3(b) shows extracted molecule templates. First, `RDF classes` are collected with their corresponding `properties`, i.e., pairs $(C_i, \mathbb{P}_i)$ are created where C_i is a class and $\mathbb{P}_i$ is a set of predicates of C_i. Then, for each RDF class C_i, object properties in $\mathbb{P}_i$ are identified. That is, RDF molecule templates describe templates of classes with properties that have this class as `rdfs:domain`. For instance, a template for `dbo:Person` contains a class `dbo:Person` and three properties `dbo:occupation`, `dbo:series`, `dbo:birthPlace`.

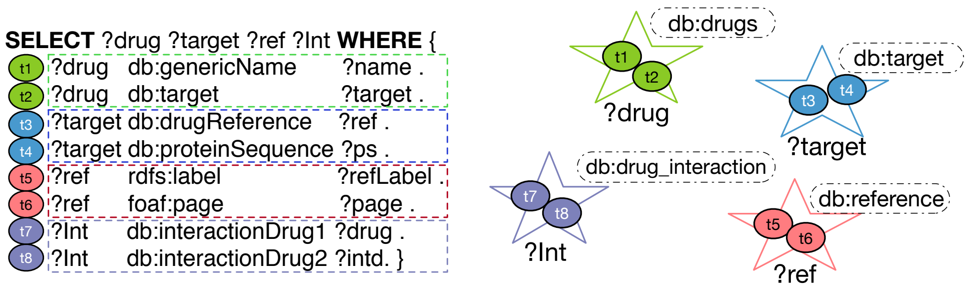
The engine then tries to link templates to each other considering `rdfs:range` value, thus, deriving a graph of sources applicable for query decomposition and source selection.

An important concept we will refer to in the subsequent Sections is a *Star-shaped Subquery (SSQ)*. The engine identifies query decompositions composed of star-shaped subqueries that match RDF-MTs, and minimize execution time and maximize answer completeness.

Star-shaped Subquery (SSQ)

Definition 6.1.2 *A star-shaped subquery* `star(S,?X)` *on a variable* `?X` *is defined as [115]:*

- `star(S,?X)` *is a triple pattern* `t={?X p o}`*, and* `p` *and* `o` *are different to* `?X`*.*
- `star(S,?X)` *is the union of two stars,* `star(S1,?X)` *and* `star(S2,?X)`*, where triple patterns in* `S1` *and* `S2` *only share the variable* `?X`*.*



```
SELECT ?drug ?target ?ref ?Int WHERE {
t1 ?drug   db:genericName      ?name .
t2 ?drug   db:target           ?target .
t3 ?target db:drugReference    ?ref .
t4 ?target db:proteinSequence ?ps .
t5 ?ref    rdfs:label          ?refLabel .
t6 ?ref    foaf:page           ?page .
t7 ?Int    db:interactionDrug1 ?drug .
t8 ?Int    db:interactionDrug2 ?intd. }
```

(a) SPARQL Query (b) Star-shaped Subqueries

Figure 6.4: **Star-shaped groups extracted from a query**. (a) SPARQL query composed of eight triple patterns that can be decomposed into four star-shapped subqueries. (b) Four star-shaped subqueries.

Fig. 6.4 shows an example of star-shaped groups identified within a query. A SPARQL query in Fig. 6.4(a) consists of eight triple patterns where each two share the same subject variable. Therefore, the query can be decomposed into four star-shaped groups and, subsequently, subqueries. The identified groups are depicted in Fig. 6.4(b). The described query engine processes the SSGs in order to derive an effective query plan and execution strategy.

We leverage the proposed federated query processing framework that provides an environment for several query planning optimizations we propose in the following Sections. Resorting to MULDER dealing with source description, query decomposition, source selection, and query execution steps, we aim to improve the query planning step defining and describing a multi-way join operator and query planner capable of employing both binary and *n*-ary join operators together within one tree plan in Section 6.2. Both approaches rely on the *star-shaped subquery* concept defined in this Section. Further, in Section 6.3 we explore benefits of combining a semantic similarity join operator presented in Section 5.3 and multi-way join operator described in Section 6.2 within one physical join operator. Implementing the proposed operators using MULDER engine we evaluate their performance in Section 6.4.

6.2 The Multi-way Join Operator for SPARQL

Existing query engines rely on physical operators, e.g., binary joins, to access and combine data required for query answering [95, 120, 121]. However, Web data sources are autonomous, and data transfer rates can considerably vary among them. Thus, operators able to complete query processing schedulers and produce results as soon as possible, are mandatory for realizing efficient query processing. State-of-the-art query engines like nLDE [121] and ANAPSID [95], rely on adaptive query strategies and symmetric binary joins, to adjust execution schedulers to fluctuating data transfer rates.

Albeit effective, symmetric binary joins can drastically impact the performance of query processing whenever source answers need to be passed through multiple operators in a query plan. Multi-way symmetric joins have been proposed to overcome this problem, and propagate and generate (intermediate) results in a single step [45] during query execution. Nevertheless, multi-way joins tailored to exploit properties of SPARQL queries, e.g., query shape, have not yet been explored in existing unified and federated query processing engines.

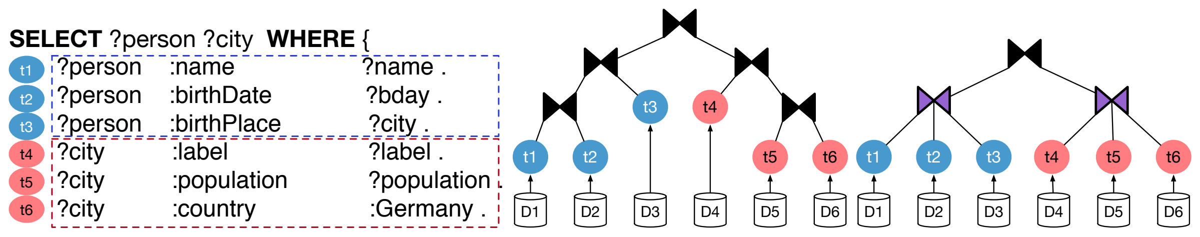
We present *SMJoin*, a multi-way join operator customized for star-shaped SPARQL queries, i.e., triple patterns that share one variable. SMJoin employs specific data structures to process intermediate results; it is controlled by a router that maintains a *dynamic probing sequence* able to reduce the amount of steps required to generate a query answer. Having an arbitrary arity, SMJoin is able to reduce the height of a query tree plan, and effectively join multiple triple patterns of a star-shaped subquery in one step. One of the main contributions of this Section is definition and description of SMJoin, a multi-way non-blocking join operator for merging results from multiple RDF data sources.

6.2.1 Motivating Example

SPARQL queries comprised of more than one triple pattern often share at least one variable. In certain queries one variable is shared among numerous triple patterns. Fig. 6.5(a) illustrates a query that consists of six triple patterns ($t_1,\dots,t_6$) that are grouped into two star-shaped subqueries, i.e., the triple patterns share the same subject variable. Moreover, each triple pattern is answered only by one source ($D_1,\dots,D_6$), respectively. Given the query and the description of sources, existing SPARQL query engines apply various query optimization strategies to construct the most efficient query plans. However, such engines employ only binary join operators in their plans. Consequently, the constructed plans, for instance, the one depicted on Fig. 6.5(b), are variations of a binary or bushy tree. That is, for a subquery around the variable `?person`, t_1 has to be joined with t_2, an intermediate result is joined with t_3. Similarly, for a `?city` subquery, t_5 has to be joined with t_6, an intermediate result with t_4. Finally, intermediate results from two subgroups are joined to obtain final answers. Overall, five joins have to be performed. Complex plans of numerous joins negatively affect query performance. Additionally, more joins produce more intermediate results that in turn increase network traffic. Despite the variety of binary join operators [45, 118, 120], there is still room for improvement.

Supporting an arbitrary amount of inputs, multi-way join operators facilitate creation of simplified query plans, produce less intermediate results, reduce network costs, and therefore, improve query performance. Fig. 6.5(c) illustrates a query plan composed of two three-way join operators, where each multi-way operator performs a join of one star-shaped group (`?person` and `?city`), respectively.

In the relational database community multi-way join operators represent an established research field [122, 126, 127]. However, to the best of our knowledge, in semantic data management and SPARQL query processing using multi-way joins remain largely unexplored. In this article, we aim to bridge this gap and propose SMJoin (SPARQL Multi-way Join), a multi-way, non-blocking join operator tailored for star-shaped groups expressed in SPARQL.



```
SELECT ?person ?city  WHERE {
t1  ?person    :name            ?name .
t2  ?person    :birthDate       ?bday .
t3  ?person    :birthPlace      ?city .
t4  ?city      :label           ?label .
t5  ?city      :population      ?population .
t6  ?city      :country         :Germany .
```

(a) SPARQL Query (b) Binary query plan (c) Multi-way query plan

Figure 6.5: **Motivating Example.** (a) SPARQL query composed of six triple patterns grouped in two star-shaped blocks; every triple pattern resides in its own source. (b) An example query plan that employs only binary join operators. (c) A query plan with multi-way joins where the plan height and number of operators are reduced.

6.2.2 The SMJoin Operator

SMJoin resembles the *MJoin* [122] and *agjoin* [95] operators, but novel enhancements are developed to evaluate star-shaped subqueries. SMJoin resorts to the background concept of *RDF Molecule Templates* (RDF-MTs) from Definition 6.1.1 to obtain from an input query such star-shaped subqueries. By Definition 6.1.2, an SSQ shares within the subgroup a variable that is instantiated by incoming tuples. Similar to Fig. 6.5(c), a multi-way operator is able to join tuples coming from different sources within one SSQ, i.e., each SSQ relates to a respective RDF-MT. On the other hand, a multi-way operator placed on higher levels of a query plan, e.g., after binary or other multi-way operators, leverages links between RDF-MTs to join multiple SSQs. That is, a multi-way operator joins SSQs and respective RDF-MTs.

Multi Join Mappings (MJMs). SMJoin utilizes Multi Join Mappings (MJMs) as inner data structures to store join intermediate results.

Multi Join Mapping (MJM)

Definition 6.2.1 *Given a star-shaped subquery $star(S, ?X)$ of triple patterns $\{tp_1, \ldots, tp_n\}$ and an instantiation $\mu(?X)$ of the join variable $?X$, an MJM is defined as a pair as follows:*

$$MJM = (\mu(?X), \{T_1, \ldots, T_n\}) \tag{6.1}$$

where T_i corresponds to a set of instantiations of the triple pattern tp_i in S such that for all the instantiations in T_i, the instantiation of the join variable $?X$ is $\mu(?X)$. In an MJM, the first argument of the pair, i.e., $\mu(?X)$, is called the MJM Head*, while the second argument, i.e., $\{T_1, \ldots, T_n\}$, is named the MJM* Tail.

Fig. 6.6 illustrates an MJM of the star-shaped subquery on the variable `?person` in Fig.6.5(a). The MJM *Head* is `:Person1` which corresponds to an instantiation of the join variable *?person*. Moreover, the second argument of the MJM is $\{T_1, T_2, T_3\}$, with the instantiations of *?name*, *?bday*, *?city* of the triple patterns tp_1, tp_2, tp_3.

MJMs are indexed by MJM Head values, i.e., an instantiation of a join variable. In Fig. 6.6, *?person* is a join variable and the MJM Head value is `:Person1`. tp_1 after evaluation returns three tuples with

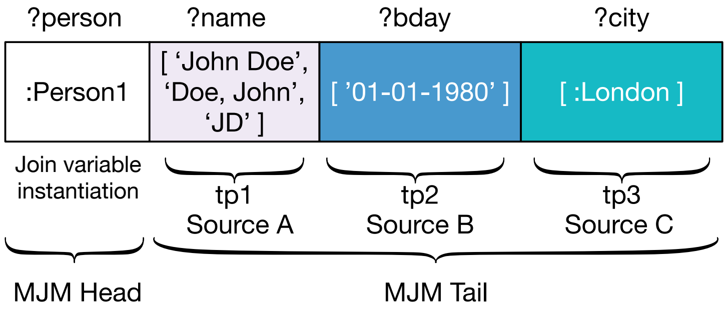


Figure 6.6: **Example of an Multi Join Mapping (MJM).** An MJM Head contains an instantiation *:Person1* of a join variable *?person*. An MJM Tail consists of instantiations of the triple patterns tp_1, tp_2, tp_3 over sources A, B, and C, respectively.

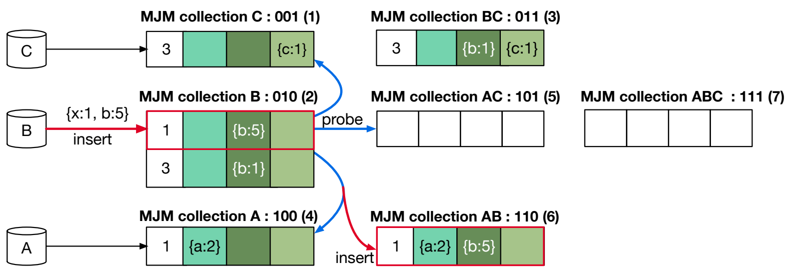
?name of `:Person1`, while tp_2 and tp_3 produce instantiations of *?bday* and *?city* for `:Person1`, respectively. Thus, the MJM *Tail* consists of three sets with the tuples collected after executing tp_1, tp_2, and tp_3 against sources A, B, and C, respectively.

Fig. 6.6 presents a complete MJM for the star-shaped subquery of `?person`, i.e., an MJM with instantiations of the triple patterns tp_1, tp_2, tp_3 for the MJM *Head* :Person1, are part of the MJM *Tail*. Once an MJM is completed, the SMJoin operator produces three results as a Cartesian product of sets in the MJM Tail, e.g., `{:Person1,'John Doe','01-01-1980',:London}`; `{:Person1, 'Doe,John', '01-01-1980', :London }`; and `{:Person1, 'JD','01-01-1980',:London}`. Referring to the plan in Fig. 6.5(c) those tuples are pushed forward as an input for a subsequent binary join operator in the query plan. In order to join multiple answers from different sources, SMJoin aims at completing MJMs for the instantiations of a given join variable. Fig. 6.7 depicts the SMJoin pipeline, i.e., from the arrival of a first tuple until the generation of the final query answer.

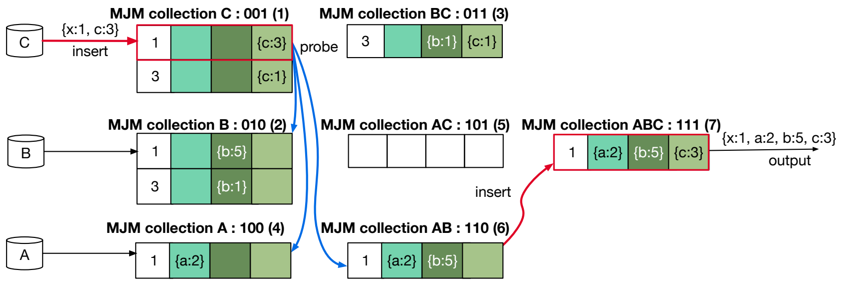
MJM Collections. An MJM collection is a set of MJMs where the *Tails* are composed of sets of instantiations of the same group of triple patterns. For example, Fig. 6.7(a) depicts seven MJM collections. In particular, the MJM collection B stores two MJMs with various instantiations (i.e., `{x:1, b:5}`, `{x:3, b:1}`) of one triple pattern (e.g., `?x :property ?b`) that was answered by the source B .

Upon initialization of the SMJoin operator with n inputs, $2^n - 1$ MJM collections are created. Instantiations received from n sources are stored in n MJM collections. Combinations of these instantiations correspond to intermediate results and are kept in $((2^n - 1) - n)$ MJM collections; only one MJM collection stores the complete results to be yielded. For instance, in Fig. 6.7(a), SMJoin is initialized with answers from sources A, B, and C; seven MJM collections are created where three MJM collections store incoming tuples from a respective source; three MJM collections accumulate intermediate results; and one MJM collection is reserved for the join results. We provide additional details how all these MJM collections are organized and ordered during a *probing sequence* in Section 6.2.3.

Computing Join Results from MJM Collections. When an input triple - an instantiation of a triple pattern - arrives from a source, several actions are triggered. Firstly, to foster results generation and reduce response time, the input tuple is probed against relevant MJMs according to a *probing sequence*, i.e., MJMs are arranged in a way to yield join results faster. In case a join match occurs when probing, an intermediate result is stored in a relevant intermediate MJM collection. If an MJM Tail for a current $\mu(?X)$ is completed, i.e., the MJM contains input tuples from all sources, then this MJM exists in the output MJM collection and is ready to yield join results. Secondly, the tuple invokes its own MJM with its $\mu(?X)$ as MJM Head and tuple value in the corresponding section in MJM Tail. Finally, the new MJM is inserted into a corresponding MJM collection, i.e., MJM collection A puts together MJMs which join


MJM collection C : 001 (1)
MJM collection BC : 011 (3)
C
3
{c:1}
3
{b:1}
{c:1}
MJM collection B : 010 (2)
MJM collection AC : 101 (5)
MJM collection ABC : 111 (7)
B
{x:1, b:5}
insert
1
{b:5}
probe
3
{b:1}
MJM collection A : 100 (4)
MJM collection AB : 110 (6)
A
1
{a:2}
insert
1
{a:2}
{b:5}

(a) Generation of Intermediate Results


MJM collection C : 001 (1)
MJM collection BC : 011 (3)
C
{x:1, c:3}
insert
1
{c:3}
probe
3
{b:1}
{c:1}
3
{c:1}
MJM collection B : 010 (2)
MJM collection AC : 101 (5)
MJM collection ABC : 111 (7)
{x:1, a:2, b:5, c:3}
output
B
1
{b:5}
1
{a:2}
{b:5}
{c:3}
3
{b:1}
insert
MJM collection A : 100 (4)
MJM collection AB : 110 (6)
A
1
{a:2}
1
{a:2}
{b:5}

(b) Generation of Query Answers

Figure 6.7: **SMJoin Intuition.** (a) Current state: join variable is x, one tuple arrived from each of the sources A, B, C, and one intermediate result resides in the MJM collection BC. Then, a tuple arrives from the source B. The newly created MJM is probed against MJM collections AC (empty), C, and A. The instantiation of the join variable x in A is the same, a new intermediate join result is inserted into the MJM collection AB. Finally, the arrived MJM is inserted into the collection B. (b) A tuple arrives from the source C. Its MJM is probed first against the MJM collection AB. A join is identified so that collections B and A are omitted. The obtained result is inserted into the MJM collection ABC and considered as final. One result tuple is produced.

variable instantiations $\mu(?X)$ contain tuples from source A.

In Fig.6.7(a), a tuple `{x:1, b:5}` arrives from the source B. Given `'x'` as a join variable, at the current state, there exist intermediate MJMs in MJM collections: A (instantiated with a value `{x:1}` in its head), B with `{x:3}`, C (`{x:3}`), and BC (instantiated with a value `{x:3}` as a join between B and C). The arrived tuple from the source B generates its MJM and is subsequently probed against three MJM collections, namely AC, A, and C. As AC is empty, the arrived MJM `{1:[{b:5}]` is probed against existing MJMs in A `{1:[{a:2}]}` and C `{3:[{c:1}]}`. $\mu(x) = 1$ occurs for a probing tuple in A; therefore, a join match is identified, a new MJM is created where the head remains the same, but the tail is a union of the tails of probed MJMs. The new intermediate MJM, i.e., `{1:[{a:2},{b:5}]}`, is inserted into the relevant MJM collection AB that accumulates joinable tuples from the sources A and B.

Furthermore, a tuple `{x:1, c:3}` arrives from the source C (cf. Fig.6.7(b)). A new MJM `{1:[{c:3}]` is created and probed against three MJM collections in the order AB, B, and A. The head of the intermediate result in AB obtained at the previous step, matches with the head of the MJM from the source C, i.e., $\mu(x) = 1$, and the join is identified. In turn, a new intermediate MJM `{1:[{a:2},{b:5},{c:3}]}` is created and inserted into the ABC MJM collection. As this MJM consists of the input tuples obtained from all three sources A, B, and C, the MJM is annotated as complete (similar to the one represented in Fig. 6.6). Thus, SMJoin is able to yield resulting joined tuples. Taking a Cartesian product of the MJM head and unique variables in the MJM tail, one tuple is pushed:

`{x:1,a:2,b:5,c:3}`; it corresponds to the outcome of a three-way SMJoin operator. Note that after probing against an MJM in *AB* and discovering a join, there is no probing against MJM collections *A* and *B* separately; thus, all their possible join permutations are already in the *AB* collection. However, if probing of an MJM from *C* against an MJM in *AB* does not produce a new intermediate result, then the collections *A* and *B* are still present in the probing sequence. SMJoin is equipped with an intra-operator router for probing sequence; further, the router is able to avoid redundant probings.

6.2.3 SMJoin Intra-Operator Router

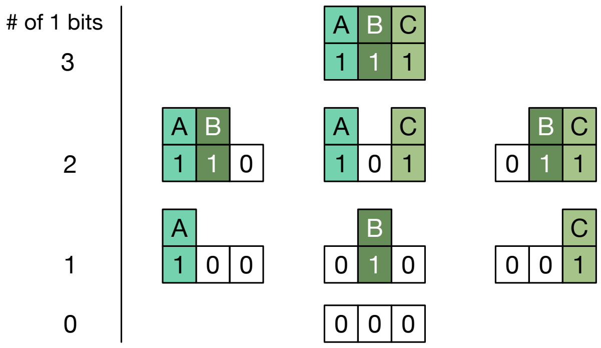
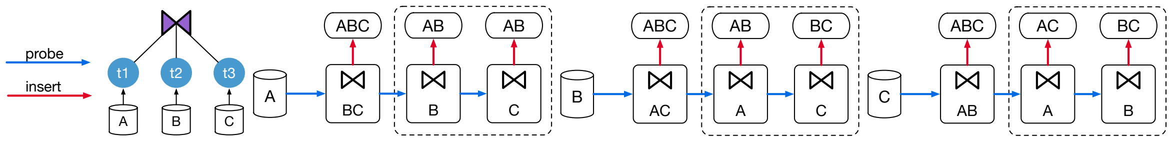


Figure 6.8: **A lattice is comprised of three sources *A*, *B*, *C* that correspond to digits in the lattice.** Total number of indices is $2^3 = 8$. The index 000 is redundant as it is not connected with any source or intermediate result. The components with indices 100, 010, 001 store incoming tuples from sources *A*, *B*, *C*, respectively. The components 110, 101, 011 store intermediate join results from *AB*, *AC*, *BC*, respectively. Index 111 denotes a component *ABC* with final SMJoin results.

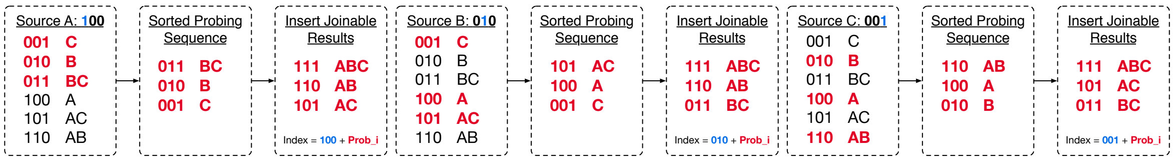
A Lattice of MJM Collections. Before describing the SMJoin router and its probing sequence generator, it is important to understand how the inner data structures, i.e., MJM collections, are organized. All the $2^n - 1$ MJM collections are ordered into a *lattice* structure (Fig. 6.8) where a binary index of an MJM collection corresponds to the contents of this MJM collection. Bit 1 at a certain position in the index number denotes sources which intermediate join results are stored in the given MJM collection. An MJM collection with index 0 becomes thus redundant as it does not contain any MJM from any source. Therefore, the overall amount of MJM collections equals to $2^n - 1$. Fig. 6.8 illustrates a lattice comprised of three sources *A*, *B*, *C*; lattice elements (MJM collection indices) consist of three digits equal to the number of sources. Indices that contain a single 1 bit (001, 010, 100) store MJMs received from the sources, i.e., MJM collection 001 (decimal index 1) accumulates MJMs from the source *C*, 010 (2) from *B*, and 100 (4) from *A*. Indices that include two 1 bits (011, 101, 110) store intermediate join results obtained from the respective sources. For instance, MJM collection with index 011 (3) consists of the intermediate join results between *B* (010) and *C* (001). Similarly, MJM collection 101 is a place for intermediate results between *A* (100) and *C* (001). Finally, index 110 shows that this MJM combines joins from *A* (100) and *B* (010). One of the natural benefits provided by a SMJoin lattice is that the index of an MJM collection to insert an intermediate result corresponds to a logical OR between two original indices, e.g., for *AB* 110 = 100 OR 010. The MJM collection with index 111 denotes eventual join operator results as three 1 bits represent that MJMs in this collection include results from all three sources *A*, *B*, and *C*. MJMs in this collection *ABC* are converted to the format of output tuples and pushed forward to the next level of a query plan.

SMJoin Probing Sequences. The SMJoin router takes advantage of the SMJoin binary indexation strategy to produce the best possible probing sequence for every attached source. An efficient probing



(a) Initialization (b) Probing sequence A (c) Probing sequence B (d) Probing sequence C

Figure 6.9: **The SMJoin Router.** (a) Initial state of SMJoin. (b) The probing sequence for the source A includes MJM collections BC, B, and C (blue arrows). Collections surrounded by a dashed line, B and C, are excluded from the probing sequence if the join occurs at the first stage when probing against BC. Successful probings are inserted into corresponding MJM collections ABC (output), AB, and AC (red arrows). (c) The probing sequence for the source B is comprised of AC, A, and C where two latter are excluded if there is a join between B and AC. (d) The probing sequence for the source C consists of AB, A, and B. If a join occurs between C and AB then A and B are excluded from the probing sequence.



(a) Probing sequence for A (b) Probing sequence for B (c) Probing sequence for C

Figure 6.10: **Computing a probing sequence.** (a) The incoming tuples from the source A are stored in the lattice component with index 100. At the first step three indices are selected as they do not contain any data from A, i.e., 001 (C), 010 (B), 011 (BC). At the second step the selected indices are sorted (desc.) by the amount of 1 bits in the binary representation, i.e., the sorted sequence is 011, 010, 001. If join is identified an index of the collection to insert an intermediate result into is computed as logical OR between source A index and probing index from the sequence, e.g., 100 OR 011 = 111. The same algorithm with selecting and sorting steps applies when computing a probing sequence for the sources B (b) and C (c).

sequence eliminates redundant comparisons between intermediate results and facilitates faster output generation. Fig. 6.9 illustrates probing sequences arranged by the SMJoin router for a given subquery of tp_1, tp_2, tp_3 joined by the SMJoin operator (cf. Fig. 6.9(a)). The Router is designed to follow two principles: *i*) Foster the join results output rate; *ii*) Eliminate redundant comparisons.

To tackle the first goal, the SMJoin router arranges a probing sequence in a specific order where the intermediate MJM collections are placed first in the queue iff they contain join results from all the sources except the incoming one. That is, given n sources, the first collection to probe is the one which MJMs already accumulate $n-1$ sources, e.g., for the source A the first collection to test is BC (cf. Fig. 6.10(a)). If a join is identified, then the MJM is directly passed to the output as the MJM contains tuples from all n sources and the insert index equals to 111 (ABC). Additionally, MJM collections B and C are removed from the probing sequence as redundant, i.e., there is no need to compare an MJM from A with B and C separately when it has already been probed against their joinable results in BC. If no join is identified at the current stage, the next MJM collection in the queue combines intermediate results from $n-2$ sources, e.g., as in Fig. 6.10(a), for the source A those are collections B and C. If a join occurs, then a new intermediate result is inserted into a corresponding collection AB or AC whereas its index is computed as a logical OR between A and B or A and C, respectively; thus lattice properties are exploited. For n sources, the probing sequence is sorted in descending order by the number of sources that contributed to a given MJM collection with intermediate results, i.e., from $n-1$ to 1. The SMJoin router organizes probing sequences B (cf. Fig. 6.10(b)) and C (cf. Fig. 6.10(c)) similarly to A in Fig. 6.10(a).

The second goal is required in high-dimensional SMJoin instantiations. Given n sources, SMJoin handles $2^n - 1$ MJM collections ($2^n - 2$ excluding the output collection). Therefore, a straightforward approach to compare an incoming MJM with the rest of MJM collections has to probe this MJM against $2^n - 3$ collections that leads to exponential time complexity. In order to reduce time complexity, the SMJoin router introduces a *dynamic probing sequence* technique that prunes a probing sequence up to one MJM collection in the best case and to $2^{n-1} - 1$ in the worst case. Fig. 6.10 illustrates how the SMJoin router identifies a probing sequence for each source. The generation of the sequence consists of two steps: *i*) Selection of the relevant MJM collections; and *ii*) Sorting the sequence.

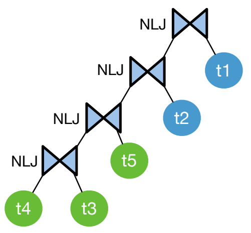
SMJoin Router Steps. At the first step, the SMJoin router selects relevant MJM collections to probe against an MJM from a given source. In terms of binary indices of a lattice, the SMJoin router selects only those indices that have a 0 bit in the position of an incoming source, i.e., a 0 bit denotes that the given source has not yet been probed against the contents of an MJM collection. Exploiting the lattice structure, given n sources, each binary index has n binary digits and there exists $2^{n-1} - 1$ indices that have a 0 bit at a certain position (excluding an empty all-zero index collection, e.g., 000 or 0000). For instance, Fig. 6.10(a) shows the probing sequence generation for the source A that is connected to its MJM collection with index 100 (4). Among all MJM collections, the SMJoin router selects only those that have a 0 bit at the third position, e.g., 001 (1) that is bound to the source C, 010 (2) that belongs to B, and 011 (3), BC, that contains join results between B and C. The collections 100 (4), 101 (5), 110 (6) are discarded as all of them contain tuples from A, and they are not applicable for probing against A.

At the second step, the SMJoin router sorts the selected indices in descending order considering the amount of 1 bits in the binary representation of an index. Each 1 bit in the binary notation refers to a source whose tuples are presented in MJMs of the collection with this index. Therefore, the SMJoin router tackles the first goal of amplifying the output rate by placing MJM collections with intermediate results of higher join *readiness* at first places in the sequence. For the source A (cf. Fig. 6.10(a)), the probing sequence after applying a sorting algorithm is arranged as 011 (3), 010 (2), 001 (1). That is, if a join occurs when probing A with BC, the obtained result is immediately pushed as an output of the SMJoin whereas collections 010 and 001 are excluded by the SMJoin router from the probing sequence. The *dynamic probing sequence* technique relies on such exclusions in order to prune redundant probings. Generally, the SMJoin router examines a binary index of a successfully probed MJM collection; it also extracts from that index all permutations of binary numbers that contain 1 bits at the same position as in the given index. The extracted numbers are excluded from the probing sequence. For example, given the index 011 (3) two number are extracted: 010 (2) as it shares bit 1 in the second position, and 001 (1) as it shares 1 bit in the first position. In complex cases, e.g., queries over five sources, if a join occurred when probing an MJM from the collection 00001 against an MJM collection with index 10110 (22), then six numbers are extracted: 10100 (20), 10010 (18), 10000 (16), 00110 (6), 00100 (4), 00010 (2). Subsequently, those numbers are excluded from the probing sequence for the collection 00001.

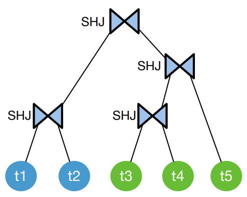
Finally, the SMJoin router identifies an index of a relevant MJM collection to insert a new intermediate result, by evaluating a logical OR between the indices of the probing collections. As illustrated in Fig. 6.10(a), an identified join between 100 (source A) and 010 (source B) is inserted into a collection with index 110 (AB) as 100 OR 010 = 110. The SMJoin router performs similarly to the described pipeline during selection, sorting, and pruning when computing a probing sequence for the source B (cf. Fig. 6.10(b)) and C (cf. Fig. 6.10(c)).

```
   SELECT DISTINCT * WHERE {
t1 ?s  dct:subject   ?o1.      #Count: 115,259,581
t2 ?s  dbo:director ?o2.      #Count: 385,773
t3 ?s  dbo:genre   dbr:Concert.       #Count: 98
t4 ?s  dbo:artist    dbr:Ana_Gabriel. #Count: 34
t5 ?s  dbo:genre   dbr:Ranchera. }   #Count: 205
```

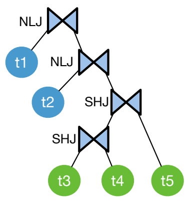
(a) SPARQL Query



(b) Nested Loop Joins (NLJ) plan



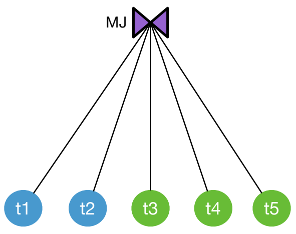
(c) Symmetric Hash Joins (SHJ) plan



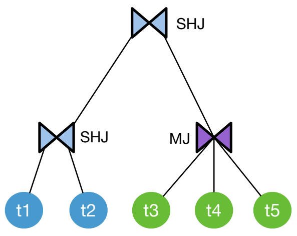
(d) SHJs + NLJs plan

Plan	ET, ms	Results
NLJs, (b)	634	12
SHJs, (c)	timeout	0
NLJs + SHJs, (d)	243	12
MJ-only, (f)	timeout	0
MJ + SHJs, (g)	timeout	0
MJ + NLJs, (h)	**195**	12

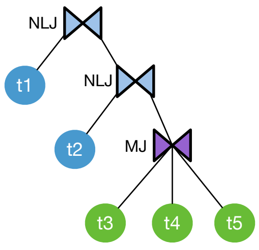
(e) Performance of binary and multi-way plans, timeout=600 sec



(f) Multi-way join (MJ) only plan



(g) MJ + SHJs plan



(h) MJ + NLJs plan

Figure 6.11: **Motivating example**. Various plans constructed and executed for a query in (a). The best result is delivered by a combination of binary and multi-way join operators.

6.2.4 On Combining Binary and Multi-way Joins

To the best of our knowledge, none of state-of-the-art SPARQL query engines employ multi-way joins nor integrate in plans with binary joins. We bridge this gap and devise an approach that incorporates best from both worlds, i.e., builds query plans combining binary and multi-way join operators in order to accelerate query execution. The SPARQL query presented in Fig. 6.11(a) comprises five triple patterns that share the same subject variable s in one star that retrieves *musical albums with their directors and subjects made by an artist Ana Gabriel in genres Ranchera and Concert*. The query is executed against the LOD-a-lot dataset [175] in the HDT format hosted on a Triple Pattern Fragments server [38].

An important feature of the query consists in drastic selectivity variance, i.e., the first two triple patterns t_1, t_2 have hundreds of thousands intermediate results whereas t_3, t_4, t_5 have less than one thousand answers. Given the query, different SPARQL engines build various query plans according to their heuristics and optimization algorithms. Fig. 6.11 illustrates some of possible planning options. State-of-the art SPARQL engines like TPF Client [38] and nLDE [121] employ binary join operators, e.g., Symmetric Hash Joins (SHJ), Nested Loop Joins (NLJ), in their plans and therefore able to construct Fig. 6.11b,c,d plans. However, there is an opportunity to utilize a Multi-way Join (MJ) operator in query planning, e.g., in a way depicted in Fig. 6.11f,g,h, which none of the existing SPARQL engines tackle. We applied TPF Client, nLDE and implemented custom multi-way query plans in order to compare the execution time of the query in Fig. 6.11(a). Fig. 6.11(e) reports the results. Clearly, plans with join operators that have to obtain all intermediate results in runtime like SHJ and MJ placed separately or together, i.e., Fig. 6.11c,f,g, are not capable of producing answers in reasonable time (timeout is set to 10 minutes). Due to low selectivity of t_1, t_2 the best option is to use nested loop joins in plans in order to avoid retrieving thousands of irrelevant intermediate results as demonstrate Fig. 6.11b,d,h. Moreover, the performance increases (at least 3x faster as shown in Fig. 6.11(e)) if a high selective subset of triple patterns is processed by symmetric operators, e.g., SHJs or MJ. Hence, the most effective plan for a given query depicted in Fig. 6.11(h) combines a multi-way join operator to join high selective triple patterns t_3, t_4, t_5 and two nested loop join operators to utilize concrete variable instantiations sent from the multi-way join in order to resolve t_1 and t_2. In this paragraph, we propose the first SPARQL query planner optimization approach that not only employs multi-way joins (blind replacement of all operators with multi-way joins is not adequate as shown above), but efficiently combines them with binary joins in order to accelerate SPARQL query execution performance.

As the first attempt to tackle this problem we utilize two heuristics: *i*) *Heuristic 1* considers a size of a star-shaped subgroup (SSG). If an SSG contains more than three triple patterns, a multi-way join will be put in the tree plan. *ii*) *Heuristic 2* relies on selectivity of triple patterns. If a query consists of more than three high-selective triple patterns that share the same join variable, a multi-way join operator is instantiated for those triple patterns. In the experimental study we obtain and analyze the first results how those heuristics contribute to the overall query performance.

In the next Section, we extend the notion of a multi-way join operator with a similarity component presented in Section 5.3.

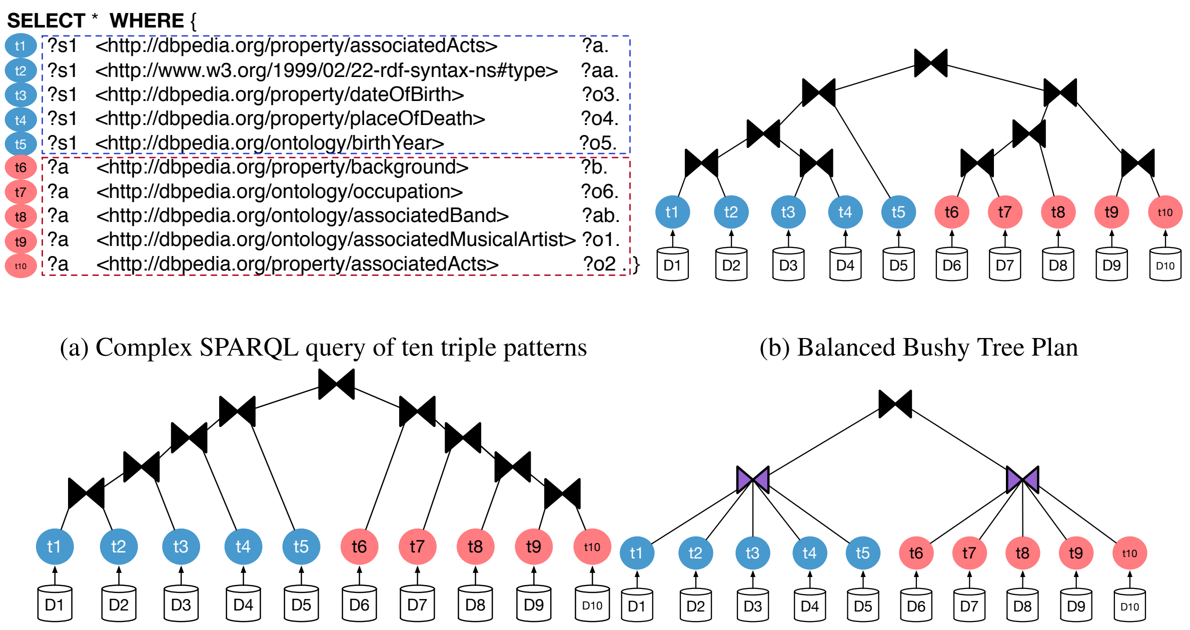
6.3 A Multi-way Similarity Join Operator

The majority of existing query engines rely on binary join-based query planners and execution methods with complexity that depends on the number of involved data sources. Moreover, traditional binary join operators are not able to distinguish between similar and different tuples, treating every incoming tuple as an independent object. Thus, if tuples are represented differently but refer to the same real-world entity, they are still considered as non-related objects. We propose MSimJoin, an approach towards a multi-way similarity join operator. MSimJoin accepts more than two inputs and is able to identify duplicates that correspond to similar entities from incoming tuples using semantic technologies. The proposed architecture describes how such components comprise a physical operator and specify inputs and outputs of the operator.

6.3.1 Motivating Example

Fig. 6.12 illustrates the intuition behind the idea of a multi-way join plan. Given a complex SPARQL query (cf. Fig. 6.12(a)) that consists of ten triple patterns and only one data source is able to answer a particular triple pattern, i.e., the query environment is federated. The task of answering the query presents severe difficulties for existing federated query engines that employ only binary join plans. For instance, some variants of bushy plans (cf. Fig. 6.12(b)) and (cf. Fig. 6.12(c)) have to perform nine joins that is computationally expensive. On the other hand, employing a multi-way join operator, i.e., 5-way as presented in Fig. 6.12(d), allows for a significant simplification of a query plan. That is, only three joins have to be performed, two multi-way joins that integrate the results of five respective triple patterns each (`t1-t5` that share a join variable `?s1` and `t6-t10` that share `?a`) and one binary join to produce a final answer of `t1` triple pattern, i.e., a join between `?s1` and `?a`.

Moreover, usually data sources are not aligned among each other and therefore contain replicated and redundant data. For instance, Fig. 6.13 illustrates the case when three data sources that answer `t8`, `t9`, `t10` of a query in Fig. 6.12(a) return entities, namely `<CCR>` and `<Creedence_Clearwater_Revival>`, that refer to the same real-world object. Traditional join operators consider tuples with such entities as completely different and thus, given that `?a` is a join variable, do not yield any output as `<CCR>` and



```
SELECT * WHERE {
t1  ?s1  <http://dbpedia.org/property/associatedActs>                ?a.
t2  ?s1  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           ?aa.
t3  ?s1  <http://dbpedia.org/property/dateOfBirth>                   ?o3.
t4  ?s1  <http://dbpedia.org/property/placeOfDeath>                  ?o4.
t5  ?s1  <http://dbpedia.org/ontology/birthYear>                     ?o5.
t6  ?a   <http://dbpedia.org/property/background>                    ?b.
t7  ?a   <http://dbpedia.org/ontology/occupation>                    ?o6.
t8  ?a   <http://dbpedia.org/ontology/associatedBand>                ?ab.
t9  ?a   <http://dbpedia.org/ontology/associatedMusicalArtist>       ?o1.
t10 ?a   <http://dbpedia.org/property/associatedActs>                ?o2 .}
```

(a) Complex SPARQL query of ten triple patterns

(b) Balanced Bushy Tree Plan

(c) Bushy Plan on Nested Loop joins

(d) Multi-way join plan

Figure 6.12: **Motivating example.** (a) A complex SPARQL query. (b) An example of a bushy tree plan. (c) An example of a nested loop joins. (d) Multi-way plan.

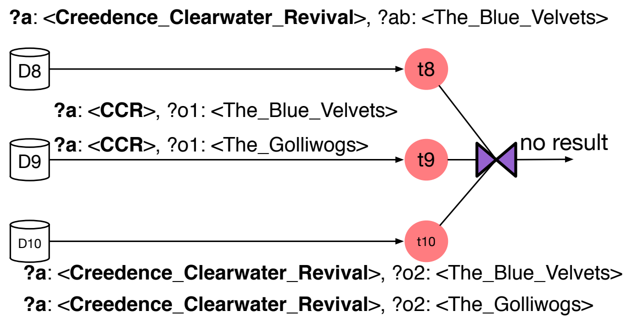
`<Creedence_Clearwater_Revival>` do no match syntactically. With the growing number of accessible data sources it is hardly ever possible to ensure that only unique entities arrive to the engine.

As stated in Definition 5.3.1, we call entities *semantically equivalent* (or *equivalent*) if they refer to the same real-world entity but are represented differently. For instance, both notions `<CCR>` and `<Creedence_Clearwater_Revival>` refer to one music band, and thus they are *semantically equivalent*. Furthermore, the task of the join operator when executing against a federation of sources is to tackle equivalent entities and produce valid results. The example of such an operator is presented in Fig. 6.13(b). The operator is able to match values of the syntactically different join variable `?a` and yields four results (obtained as a Cartesian product of non-join variable values `?ab`, `?o1`, `?o2`) whereas a non-similarity operator is not able to yield any result (cf. Fig. 6.13(a)). This Section presents an approach towards such a join operator that combines both features, that is, support for numerous inputs and semantic similarity mechanisms.

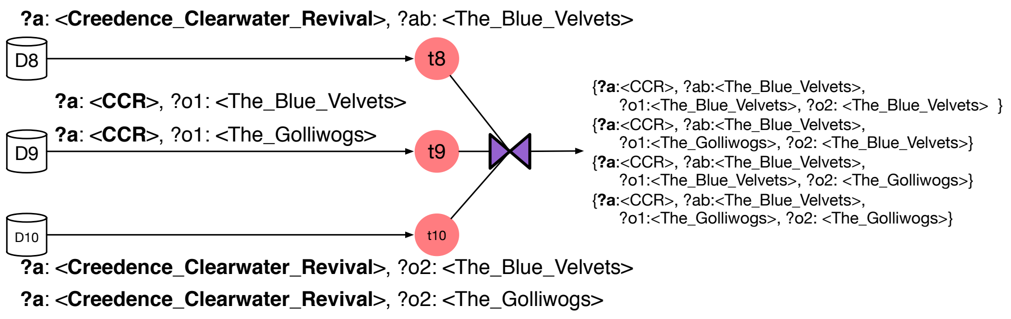
6.3.2 The MSimJoin Operator

SPARQL provides standardized means for querying federations of data sources [176]. In this work, the problems of query rewriting and optimal source selection are out of the scope, we thus resort to existing approaches, e.g.,[115]. We assume the data sources are identified and properly queried so that every source returns a stream of tuples to be joined by a join variable with other streams. Therefore, the input of a *n*-way join operator consists of *n* input streams.

We employing the *SMJoin* operator defined in Section 6.2 that accepts more than two such streams and produces a join only if an instantiation of a join variable is shared in tuples among all sources. Fig. 6.14 illustrates the intuition behind the proposed similarity join operator that follows the *SJoin*


?a: <Creedence_Clearwater_Revival>, ?ab: <The_Blue_Velvets>
D8
t8
?a: <CCR>, ?o1: <The_Blue_Velvets>
?a: <CCR>, ?o1: <The_Golliwogs>
D9
t9
no result
D10
t10
?a: <Creedence_Clearwater_Revival>, ?o2: <The_Blue_Velvets>
?a: <Creedence_Clearwater_Revival>, ?o2: <The_Golliwogs>

(a) Non-similarity operator

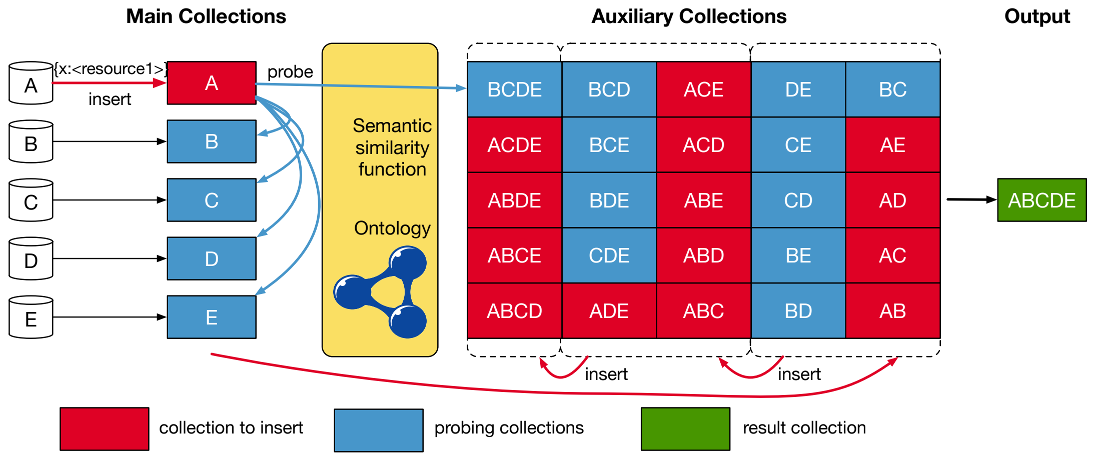

?a: <Creedence_Clearwater_Revival>, ?ab: <The_Blue_Velvets>
D8
t8
?a: <CCR>, ?o1: <The_Blue_Velvets>
?a: <CCR>, ?o1: <The_Golliwogs>
D9
t9
D10
t10
{?a:<CCR>, ?ab:<The_Blue_Velvets>, ?o1:<The_Blue_Velvets>, ?o2: <The_Blue_Velvets> }
{?a:<CCR>, ?ab:<The_Blue_Velvets>, ?o1:<The_Golliwogs>, ?o2: <The_Blue_Velvets>}
{?a:<CCR>, ?ab:<The_Blue_Velvets>, ?o1:<The_Blue_Velvets>, ?o2: <The_Golliwogs>}
{?a:<CCR>, ?ab:<The_Blue_Velvets>, ?o1:<The_Golliwogs>, ?o2: <The_Golliwogs>}
?a: <Creedence_Clearwater_Revival>, ?o2: <The_Blue_Velvets>
?a: <Creedence_Clearwater_Revival>, ?o2: <The_Golliwogs>

(b) Semantic similarity operator

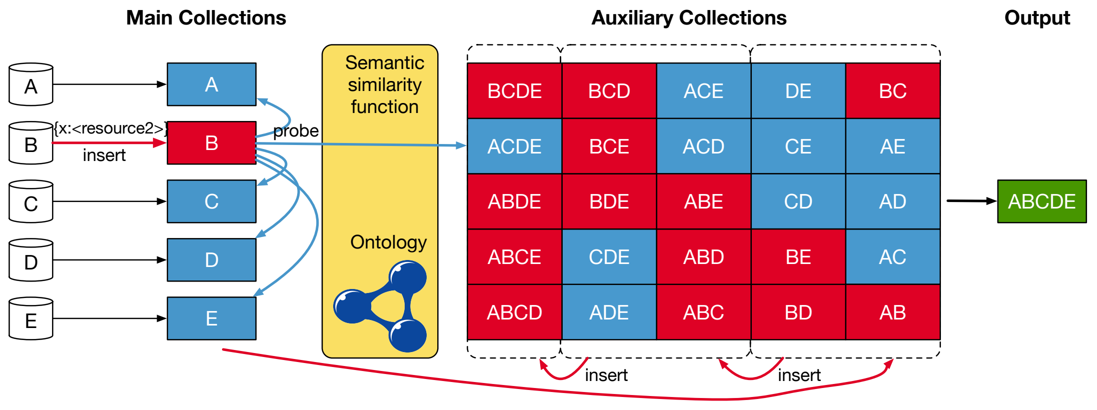
Figure 6.13: **Motivating Example (continued).** (a) A non-similarity join operator is not able to identify semantically equivalent entities and therefore produces zero results. (b) A semantic similarity operator matches `<CCR>` with `<Creedence_Clearwater_Revival>` and produces four tuples.

approach defined in Section 5.3. In order to apply semantic similarity techniques, input tuples are converted to the format of *RDF molecules*, cf. Definition 5.2.1. Given that the original query is written in SPARQL, RDF molecules can be constructed using query predicates, e.g., the returned tuple from the data source `?a: <CCR>, ?o1: <The_Golliwogs>` is converted to an RDF triple `<CCR> dbp:associatedMusicalArtist <The_Golliwogs>`.

The operator employs and extends the classical *probe-insert* methodology to support an arbitrary arity and store intermediate results. For instance, as depicted in Fig. 6.14(a), a tuple arrives from the source A. It is subsequently converted to an RDF molecule and probed against respective collections of intermediate results. To produce results as soon as they are ready, a molecule is probed against intermediate results in a specific order, i.e., at first, against a collection *BCDE* that might contain join results arrived before from B, C, D, and E sources, respectively, i.e., of $n-1$ length. If the tuple from A and a tuple from *BCDE* share the same join variable instantiation, then the join is found and the result is put into the output collection *ABCDE*. If the join condition is not met, the molecule is probed against collections of $n-2$ length that do not contain tuples from A, namely *BCD*, *BCE*, *BDE*, *CDE*. Contrary, if the join condition is satisfied, then a new intermediate result is inserted into a respective collection, i.e., *ABCD*, *ABCE*, *ABDE*, *ACDE*. Otherwise, the molecule is probed against collections of $n-3$ length that combine intermediate results from two sources according to Fig. 6.14. Finally, if all intermediate collections did not satisfy a join condition, the molecule is compared against main collections that accumulate direct RDF molecules from sources B, C, D, E. Any yielded intermediate results are inserted into *AB*, *AC*, *AD*,



(a) A `resource1` arrives from A. The tuple is probed against the auxiliary tables that do not contain A of length four (BCDE), three (BCD, BCE, etc), two (DE, CE, CD, BE, etc), and against the main collections (B, C, D, E).



(b) A `resource2` arrives from B. The tuple is probed against the auxiliary tables that do not contain B of length four (ACDE), three (CDE, ADE, etc), two (DE, CE, CD, AE, etc), and against the main collections (A, C, D, E).

Figure 6.14: **Multi-way similarity join intuition.** A resource tuple arrives from one of the source datasets. Auxiliary collections contain intermediate join result obtained from respective sources. The incoming the tuple is semantically probed against a set of auxiliary and main collections. The completed joinable result is put in the output collection (ABCDE). New intermediate results are inserted into a respective collection, e.g., a join between A and BC is stored in ABC.

AE, respectively. The pipeline for a tuple arrived from the source *A* is depicted in Fig. 6.14(a). A similar pipeline for a tuple arrived from *B* is depicted in Fig. 6.14(b). The main difference for the source *B* is in the new probing collections that do not contain *B* and inserts into respective collections with *B*, e.g., the intermediate join among *B* and *CDE* is inserted into *BCDE*.

Moreover, a distinctive feature of the approach consists in the probing mechanism. While traditional join operators search for a full syntactic matching of a join variable instantiation, a similarity join operator performs a semantic comparison of the probed RDF molecules. The operator resorts to semantic similarity functions, for example, *GADES* [108], that are able to deduce a numerical value of relatedness

Table 6.1: The *MSimJoin* Time Complexity. Results for the steps of Hashing, Probing, and Matching where n is operator arity, k is the number of incoming tuples from each input, $n \ll k$.

Stage	**Best Complexity**	**Average Complexity**	**Absolute Worst-case Complexity**
Hashing	$O(k \cdot n)$	$O(k \cdot n)$	$O(k \cdot n)$
Probing	$O(k \cdot n)$	$O(k \cdot n^2 \log n) \xrightarrow[n \ll k]{} O(k \cdot n)$	$O(k \cdot n \cdot 2^n) \xrightarrow[n \ll k]{} O(k \cdot n)$
Matching	$O(sim_f)$	$O(sim_f)$	$O(sim_f)$
Overall	$O(k \cdot n) \cdot O(sim_f)$	$O(k \cdot n) \cdot O(sim_f)$	$O(k \cdot n) \cdot O(sim_f)$

of two molecules given an ontology. An ontology provides additional knowledge and facts, logical axioms, class hierarchies necessary for computation of a similarity score. If the resulting similarity value exceeds a certain threshold then the molecules are considered similar and the join operator processes the given join variable instantiations as the same. For instance, a semantic similarity function is able to compute a high score for `<CCR>` and `<Creedence_Clearwater_Revival>` resources. Therefore, the join operator is able to identify a join among the molecules that contain such instantiations and yield a complete result as aimed in the example in Fig. 6.13(b). The threshold value for a semantic similarity function might be chosen either manually, i.e., after analyzing the distribution of similarity scores in order to keep the balance between number of possible candidates and their overall relatedness, or might be learned automatically if the function resorts to machine learning algorithms [177].

6.3.3 Query Plan Tree Height Reduction

Managing arbitrary arity allows the query optimizer to identify tree plans of lower height. This property of MSimJoin-based plans is illustrated in Fig. 6.13. MSimJoin joins triple patterns that share the same join variable and pushes the result further to the tree plan. At higher levels binary joins still have to be performed. Therefore, Y MSimJoin operators are considered as Y inputs for binary joins which leads to the following tree height reduction. Given Y MSimJoin operators of arity in range $[n; k]$ in a query plan P, the height of P converges to $\lceil log_2(Y) + 1 \rceil$ in the best case, i.e., comprising a bushy tree plan of MSimJoins, and to Y in the worst case of a left-linear plan. The resulting height h is thus lying in the range $\lceil log_2(Y) + 1 \rceil \leq h \leq Y$. Note that arity of MSimJoin operators does not affect the resulting height. For instance, two five-way MSimJoins in Fig. 6.12(d) comprise a bushy plan of height $h = \lceil log_2 2 + 1 \rceil = 2$.

On the other hand, binary joins are not able to complete the join of Y $[n; k]$ sources within one step. Therefore, for binary join plans there exists an additional sub-tree that increases the tree plan height, i.e., the sub-tree of $[n; k]$ sources that share the same join variable. In the best case of a bushy tree plan (cf. Fig. 6.12(b)) the additional height h_a converges to $h_a = \lceil log_2(n) \rceil$. In the worst case of a left linear plan the additional height converges to $h_a = k - 1$ as depicted in Fig. 6.12(c). Having in mind that the first layer when computing MSimJoin tree height is already included in h_a the overall height of a binary plan tree equals to $h = \lceil log_2(n) \rceil + \lceil log_2(Y) \rceil$ in the best case and $h = k + Y - 2$ in the worst case when both additional and main components comprise a left linear join plan. For instance, a bushy tree-like variant of a plan in Fig. 6.12(b) has a total height of $\lceil log_2 5 \rceil + \lceil log_2 2 \rceil = 4$, whereas the height of another bushy tree-like option with a left linear additional component in Fig. 6.12(c) equals to $h = 5 - 1 + \lceil log_2 2 \rceil = 5$.

Clearly, application of MSimJoin is able to reduce the query plan tree height.

6.3.4 Time Complexity Analysis

The MSimJoin operator of arity n receives k tuples from each input (for the sake of simplicity, we assume each source sends equal amount of tuples). We consider realistic querying scenarios $n \ll k$, i.e., a number of incoming tuples is always orders of magnitude bigger than a number of inputs. To estimate the complexity, three stages have to be analyzed, i.e., hashing stage that includes the *insert* step of the operator, probing stage that involves a probing sequence, and matching stage that performs semantic comparison of given tuples. We analyze the best, average, and worst cases of the operator complexity that depend largely on the complexity of the probing stage. The findings are presented in Table 6.1.

As MSimJoin builds upon traditional hash join approaches, the complexity of the first stage depends on a number of incoming tuples. In the *probe-insert* methodology, each of kn tuples has to be hashed and inserted (insert in a hash table is always done in $O(1)$) into a corresponding hash table which asymptotically converges to $O(k \cdot n)$ in all cases.

The probing stage employs the router which behavior we explained in Section 6.2.3. The complexity of probing depends on a probing sequence derived by the router, probability of hash match among intermediate collections, and position of the collection in the sequence that in turn affects pruning. As discussed above, in the worst case when the operator contains a lot of intermediate results collections but with low chance of producing a full answer, the router is not able to prune the probing sequence efficiently. Therefore, kn incoming tuples have to be compared against $2^{n-1} - 1$ collections. As hash lookup is completed in $O(1)$, the absolute worst-case complexity converges to $O(k \cdot n \cdot 2^n)$ exhibiting exponential law. However, as $n \ll k$ the exponential factor 2^n converges to a constant and hence can be discarded from the asymptotic approximation. Given that, the probing complexity converges to $O(k \cdot n)$. Whereas the worst case is indeed rare, on average, the router is able to prune the probing sequence to $n \log n$ collections for probing with kn tuples. Again, due to $n \ll k$ the quasilinear factor $n \log n$ converges to a constant. Hence, the average complexity is $O(k \cdot n)$. In the best probing case each incoming tuple has a join with the first element in the probing sequence thus pruning out all other collections. That is, each tuple is probed by the router only once. This corresponds to $O(k \cdot n)$ complexity.

The complexity of the matching stage distinguishes MSimJoin from other hash join operators. Traditional operators compare tuples seeking for exact syntactical matching of join variable instantiations which can be performed in $O(1)$ whereas MSimJoin resorts to semantic similarity functions sim_f seeking *semantic equivalence*. Thus, the complexity of a semantic similarity function $O(sim_f)$ defines the complexity of the matching stage.

Analyzing overall complexity of MSimJoin we notice that matching happens while probing. Hence, their complexities have to be chained and multiplied. Total asymptotic time complexity of MSimJoin when $n \ll k$ is therefore equal for the three following cases:

- $O(k \cdot n \cdot 2^n) \cdot O(sim_f) \xrightarrow[n \ll k]{} O(k \cdot n) \cdot O(sim_f)$ in the worst case;
- $O(k \cdot n^2 \log n) \cdot O(sim_f) \xrightarrow[n \ll k]{} O(k \cdot n) \cdot O(sim_f)$ on average;
- $O(k \cdot n) \cdot O(sim_f)$ in the best case.

In the majority of use-cases MSimJoin exhibits polynomial complexity that additionally depends on the complexity of a semantic similarity function. If $O(sim_f)$ is polynomial as well, the whole operator converges in polynomial time. Moreover, if MSimJoin is simplified to two inputs, the best case complexity $O(2k) \cdot O(sim_f)$ converges to the traditional hash join complexity with additional sim_f component.

In this Section, we presented several approaches for optimizing query planning and execution. The SMJoin operator described in Section 6.2 allows to join multiple inputs simultaneously. We then devised a query planner able to employ both binary and multi-way join operators in building query tree plans. We would like to emphasize that those techniques are the first of their kind in semantic data management, i.e., a general-purpose multi-way join operator and query planner supporting the operator were not explored by the community previously. Finally, in Section 6.3 we showed MSimJoin, a physical operator that combines advantages of a semantic join operator and multi-way join operator able to join semantically equivalent entities coming from multiple sources. In the following Section, we present experimental results of MULDER, the federated query engine that provides an environment for our operators and planner. Then, we evaluate the operators and conclude answering the **RQ3**.

6.4 Experimental Study

6.4.1 Evaluating MULDER

We empirically study the efficiency and effectiveness of MULDER to identify query decompositions and select relevant data sources. In the first experiment, we assess the query performance of MULDER utilizing RDF molecule templates and templates generated using the METIS [178] and SemEP [179] partitioning algorithms. In a second experiment, we compare MULDER with the federated query engines ANAPSID and FedX for two well-established benchmarks – BSBM and FedBench. We address the following research questions: **Q**1) Do different MULDER source descriptions impact on query processing in terms of efficiency and effectiveness? **Q**2) Are RDF Molecule Template based query processing techniques able to enhance query execution time and completeness?
Benchmarks and Queries: *i*) *BSBM*: We use the synthetic Berlin SPARQL Benchmark to generate 12 queries (with 20 instantiations each) over a generated dataset containing 200 million triples. *ii*) *FedBench*: We run 25 FedBench queries including cross-domain queries (CD), linked data queries (LD), and life science queries (LS). Additionally, 10 complex queries (C) proposed by [115] are included. The queries are executed against the FedBench datasets. A SPARQL endpoint able to access a unified view of all the FedBench datasets serves as gold standard and baseline.
Metrics: *i*) `Execution Time`: Elapsed time between the submission of a query to an engine and the delivery of the answers. Timeout is set to 300 seconds. *ii*) `Cardinality`: Number of answers returned by the query. *iii*) `Completeness`: Query result percentage with respect to the answers produced by the unified SPARQL endpoint with the union of all datasets.
Implementation: The MULDER[3] decomposition and source selection, and query planning components are implemented in Python 2.7.10, and integrated into ANAPSID [95], i.e., MULDER plans are executed using ANAPSID physical operators. Experiments are executed on two Dell PowerEdge R805 servers, AMD Opteron 2.4GHz CPU, 64 cores, 256GB RAM. FedBench and BSBM datasets are deployed on one machine as SPARQL endpoints using *Virtuoso 6.01.3127*, where each dataset resides in a dedicated Virtuoso docker container.

Experiment 1: Comparison of MULDER Source Descriptions

We study the impact of MULDER RDF-MTs on query processing, and compare the effect of computing molecule templates using existing graph partitioning methods: METIS and SemEP. We name MULDER-SemEP and MULDER-METIS, the version of MULDER where molecule templates have been computed using SemEP and METIS, respectively. Co-occurrences of predicates in the RDF triples of a dataset *D*

[3] `https://github.com/EIS-Bonn/MULDER`

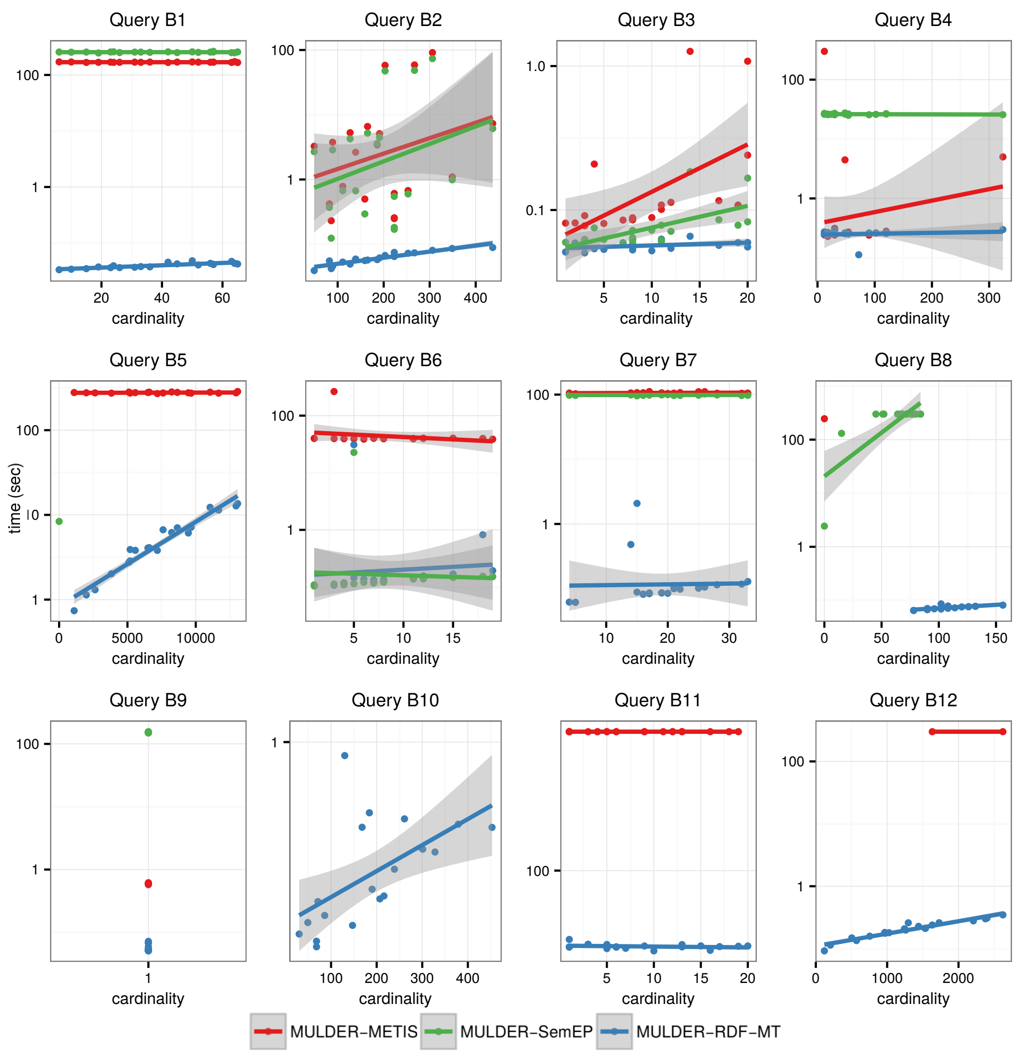


Figure 6.15: **BSBM: Performance for MULDER source descriptions.** RDF molecules are computed using: RDF-MTs, SemEP, and METIS. RDF-MTs allow MULDER to identify query decompositions and plans that speed up query processing by up to two orders of magnitude, without affecting completeness.

are computed. Given predicates *p* and *q* in *D*, co-occurrence of *p* and *q* (*co(p,q,D)*) is defined as follows:

$$co(p,q,D) = \frac{|subject(p,D) \cap subject(q,D)|}{|subject(p,D) \cup subject(q,D)|} \tag{6.2}$$

Where *subject(q,D)* corresponds to the set of different subjects of *q* in *D*. A graph GP_D where nodes correspond to predicates of *D* and edges are annotated with co-occurrence values is created, and given as input to SemEP and METIS. The number of communities determined by SemEP is used as input in METIS. METIS and SemEP molecule templates are composed of predicates with similar co-occurrence values. Each predicate is assigned to only one community.

Fig. 6.15 reports on execution time and answer cardinality of BSBM queries. The observed results suggest that knowledge encoded in RDF-MTs allows MULDER to identify query decompositions and plans that speed up query processing by up to two orders of magnitude, while answer completeness is not

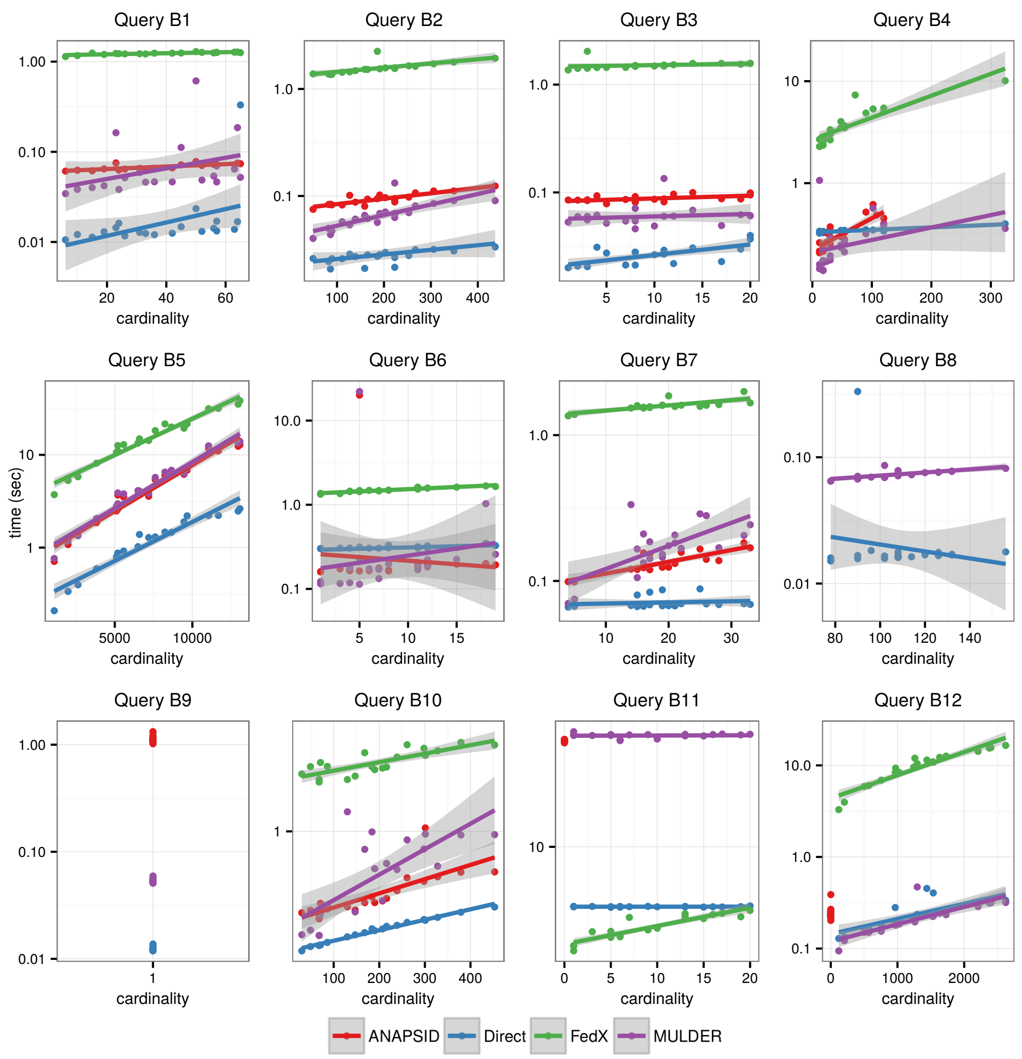


Figure 6.16: **BSBM: Performance of Federated Engines.** MULDER and ANAPSID outperform FedX in terms of query execution time, while MULDER overcomes ANAPSID in terms of completeness. Direct represents a unified SPARQL endpoint over one dataset with all the federation RDF triples.

affected. Specifically, MULDER-RDF-MTs is able to place in SSQs non-selective triple patterns, while MULDER-SemEP and MULDER-METIS group non-selective triple patterns alone in subqueries. Thus, size of intermediate results is larger in MULDER-SemEP and MULDER-METIS plans, negatively impacting execution time. Hence, MULDER-RDF-MTs outperform state-of-the-art algorithms.

Experiment 2: Comparison of Federated Query Engines

Performance of BSBM Queries. Fig. 6.16 reports on the throughput of the federated engines ANAPSID, FedX, and MULDER for all BSBM queries. In many queries, MULDER and ANAPSID exhibit similar query execution times. FedX is slower by at least one order of magnitude. ANAPSID returns query answers fast but at the cost of completeness, as can be observed in the queries B4, B7, B11, and B12. In addition, FedX and ANAPSID fail to answer B8 which is completely answered by MULDER.

Performance of FedBench Queries. Fig. 6.17 visualizes the results of the four FedBench groups of

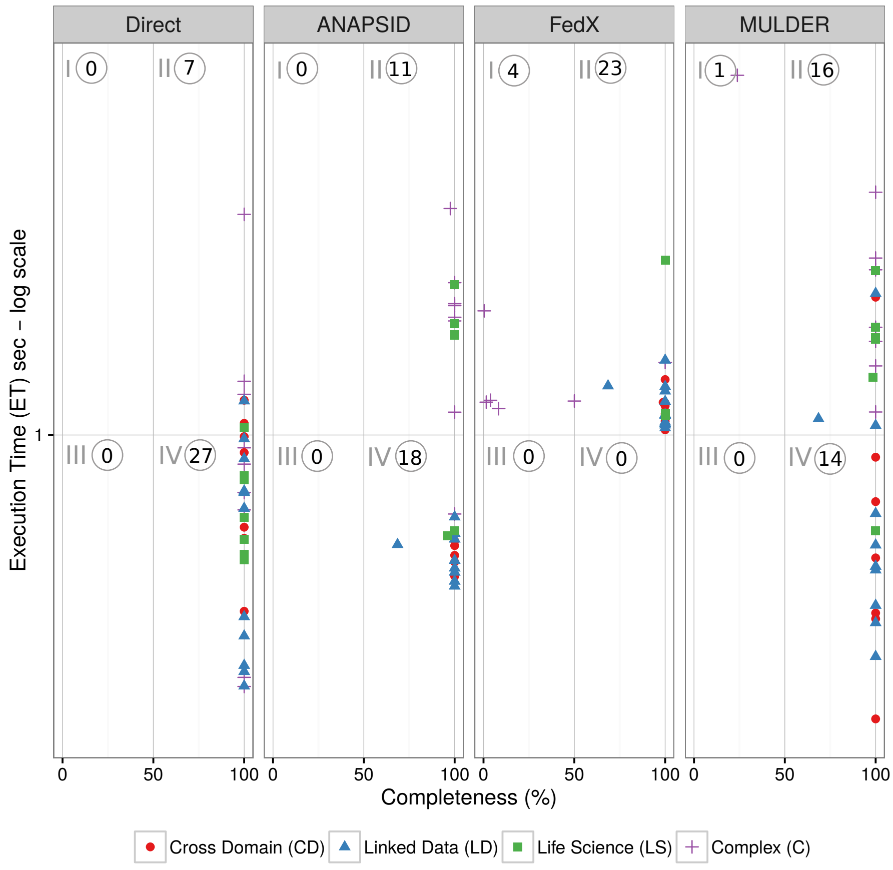


Figure 6.17: **FedBench: Execution Time and Completeness of Federated Engines.** ANAPSID, FedX and MULDER manage to answer 29, 27, and 31 queries, respectively. Direct represents a unified SPARQL endpoint that is able to answer 34 of the 35 benchmark queries before timing out.

queries (CD, LD, LS, C) in terms of answer completeness and query execution time. Measurements that are located in Quadrants I and III indicate bad performance and incomplete results, points in Quadrant IV are the best in terms of execution time and completeness, i.e., they correspond to a solution to the query decomposition problem; finally, points in Quadrant II show complete results but slower execution times. MULDER outperforms ANAPSID and FedX with regard to the number of queries it manages to answer: ANAPSID answers 29, FedX 27, and MULDER 31 out of 35 queries (Query C9 could not be answered by any of the engines). In particular, MULDER delivers answers to queries C1, C3, C4, LS4, LS5, and LS6 for which FedX fails and CD6 and LD6 for which ANAPSID fails. FedX returns complete and partially complete results for 20 and 7 queries respectively, exhibiting high execution times though (>1s). In comparison to ANAPSID, MULDER achieves in general higher completeness of results, but at the cost of query execution time. For instance, C2, C8, and LD1 are answered by ANAPSID faster by almost one order for magnitude.

Results observed in both benchmarks allow us to positively answer **Q1** and **Q2**, and conclude that RDF-MTs indeed enable MULDER decomposition and planning methods to identify efficient and effective query plans. Thus, as a novel federated query engine technique, MULDER provides a solid foundation for tailoring particular components of the query processing workflow. In the following paragraph, we investigate whether multi-way joins are able to improve query planning and execution.

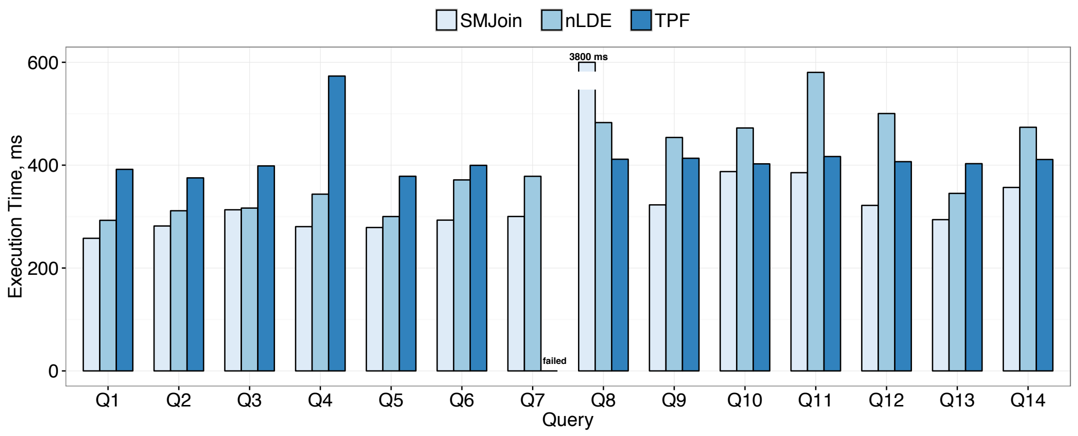


Figure 6.18: **Benchmark 1. High selective queries.** SMJoin demonstrates the best performance in all queries except Q8. Q8 takes almost four seconds as all its triple patterns are less selective compared to other queries. Q7 is not answered by TPF.

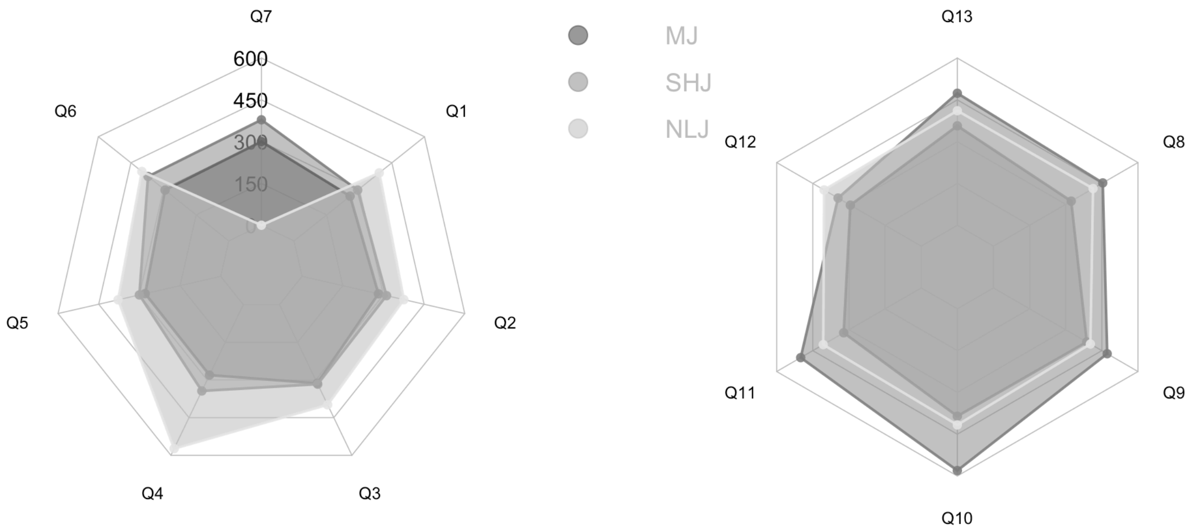
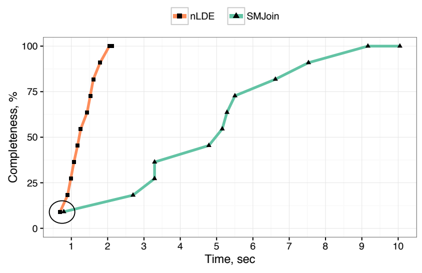


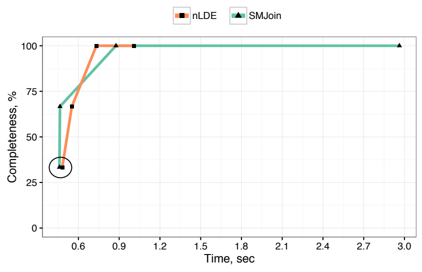
Figure 6.19: **Experiment 1. High-selective queries.** Total query execution time (ms). MJ outperforms SHJ and NLJ in all queries. Note that SHJ is faster than NLJ in Q1-Q7 and, in turn, NLJ is faster then SHJ in Q8-Q13.

6.4.2 Evaluating SMJoin

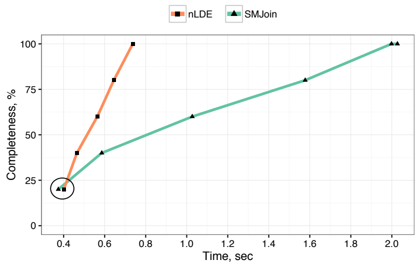
We empirically assess the performance of the SMJoin operator and benefits of a multi-way join setup compared to state-of-the-art SPARQL query processing engines, namely nLDE (Network of Linked Data Eddies) [121] and Triple Pattern Fragments Client [38]. The experimental configuration is as follows:
Benchmark and Queries: We applied two benchmarks of queries. Benchmark 1 contains 14 queries of high selectivity. Each query is composed of up to five triple patterns that share the same join variable, i.e., one star-shaped group is evaluated by a five-way SMJoin. Each triple pattern returns less than 100 intermediate results. Benchmark 2 consists of four queries that exhibit lesser selectivity, i.e., the number of intermediate results is two orders of magnitude larger than in Benchmark 1. Each query includes up to four triple patterns, i.e., a four-way SMJoin is invoked. In order to evaluate only the SMJoin operator performance not affecting query planning, all queries include one star-shaped subquery. Overall, 18



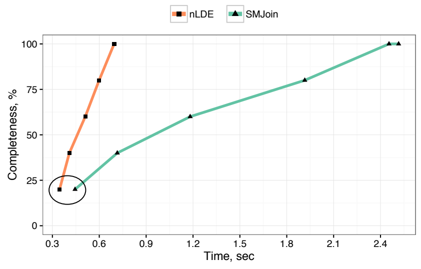
(a) LS Q1



(b) LS Q2



(c) LS Q3



(d) LS Q4

Figure 6.20: **Benchmark 2. Low-selective queries.** Completeness over time. SMJoin yields the first answer at about the same time as nLDE. However, due to lesser selectivity SMJoin has to process more intermediate results, and therefore, it finishes later. SMJoin takes more time to produce complete results in Q1(a), Q3(c), Q4(d) sending triple patterns independently whereas nLDE employs Nested Loop Join. In Q2(b) SMJoin produces complete results at the same rate as nLDE but has to process of all intermediate results that takes two additional seconds.

queries[4] are executed five times, an average value is reported.
Dataset: English version of DBpedia 2015 is converted into HDT format and hosted on top of the TPF Server. The total number of triples in the dataset is 837,257,959.
Metrics: *i*) *Execution Time*: Elapsed time between the submission of the query to an engine and the arrival of all the answers. Time corresponds to absolute wall-clock system time as reported by Python `time.time()`. Timeout limit is set to 600 seconds. *ii*) *Completeness*: Percentage of answers produced by a query engine when executing a query compared to the number of final query results.
Implementation: SMJoin operator is implemented in Python 2.7.10. We also created a decomposer of SPARQL Basic Graph Patterns (BGPs) into star-shaped subqueries. The query decomposer invokes SMJoin for each star-shaped subquery, and each triple pattern of the subquery is sent independently to a Triple Pattern Fragment (TPF) Server. For example, for a query composed of five triple patterns, five independent queries are posed. The experiments were executed on a Ubuntu 16.04 (64 bits) Dell PowerEdge R805 server, AMD Opteron 2.4 GHz CPU, 64 cores, 256 GB RAM.

Experiment 1: High Selective Queries. Experiment 1 evaluates the SMJoin performance on the set of high selective queries from the Benchmark 1. Fig. 6.18 presents the observed results against 14 queries compared to nLDE and TPF engines. Each query is executed five times and the arithmetic mean is reported. The observed results show that SMJoin outperforms on average these two state-of-the-art engines in 13 queries. This suggests that one-stage multi-way (i.e., three- to five-way) joins outperform binary join plans for very selective queries composed of very selective triple patterns, i.e., triple patterns

[4] `https://github.com/migalkin/SMJoin-experiments/tree/master/queries/new_queries`

with one variable and around 100 instantiations. nLDE and TPF generally take in average more time to execute very selective queries compared to SMJoin, but vary in performance pairwise. For example, nLDE is faster than TPF in Q1-Q7, Q13 (Q7 is not answered by TPF and its time marked as zero), whereas TPF outperforms nLDE in Q8-Q12, Q14. SMJoin is less effective in Q8 producing complete results in about four seconds; this behavior is explained by the low selectivity of the query triple patterns. In SMJoin, triple patterns are posed independently; thus, the more results the TPF server returns, the higher the time SMJoin waits until all tuples arrive and joins are performed.

Experiment 2: Low-Selective Queries. We assess the continuous SMJoin performance on low-selective queries from the Benchmark 2. Fig. 6.20 depicts the generation of query answers over time. SMJoin and nLDE are compared because as adaptive query processing techniques, they are able to produce results incrementally. Despite the higher amount of intermediate results, SMJoin is able to yield the first result at about the same time as nLDE. It is important to note that nLDE exploits information about the cardinality of the triple pattern fragments, and evaluates first the most selective patterns; nLDE also opportunistically places Nested Loop Joins to produce instantiations for low selective triple pattern fragments. In consequence, nLDE is able to drastically reduce the amount of intermediate results, and speed up execution time. In contrast, SMJoin works under the assumption of zero knowledge about the cardinality of the triple pattern fragments, and sends every triple pattern to the TPF server independently. These two different approaches of scheduling query execution allow for explaining why nLDE is faster than SMJoin over time. This is particularly important in queries like Q1 (cf. Fig. 6.20(a)), Q3 (cf. Fig. 6.20(c)), and Q4 (cf. Fig. 6.20(d)). Fig. 6.20(b) shows the impact of a large number of intermediate results on the SMJoin performance. Thus, the complete query results are produced during the first 0.9 seconds. This is comparable to nLDE; however, SMJoin has to finish processing of all tuples received from the TPF server, thus taking two more seconds are produced. **Discussion.** Observed results from Experiments 1 and 2 suggest that selecting SMJoin for high selective triple patterns and binary join operators for low-selective patterns allow for efficient execution of SPARQL query. To generate these plans, existing query decomposition and optimization techniques required to be extended to consider both join operators of different arity, i.e., binary and multi-way joins, as well as cardinality of triple pattern fragments.

6.4.3 Evaluating Combinations of Binary and Multi-way Operators

We empirically assess the performance of the approach and benefits of combining binary and multi-way operators in one query plan compared to the state-of-the-art SPARQL query processing engine Triple Pattern Fragments Client [38]. The experimental configuration is as follows:
Benchmark and Queries: We applied a benchmark that consists of 10 high-selective queries from domains of Music, Movies, People, and Places. Each query is composed of up to seven triple patterns that share the same join variable, i.e., one star-shaped group can be evaluated by a multi-way join operator or a series of binary joins. Each triple pattern returns less than 100 intermediate results. In order to evaluate only the join operators performance not affecting query planning, all queries include one star-shaped subquery. Overall, 10 queries are executed five times, an average value is reported.
Dataset: We employ LOD-a-lot [175] in the experiments as the largest available RDF dataset that consists of 28 billion triples compressed into HDT format [180]. The compressed dataset size is 524 GB and requires at least 16 GB RAM for memory footprint. **Metrics:** *i*) *Execution Time*: Elapsed time between the submission of the query to an engine and the arrival of all the answers. Time corresponds to absolute wall-clock system time as reported by Python `time.time()`. We report ET^{-1} time that corresponds to multiplicative inverse of Execution Time ET. *ii*) *Completeness (Comp)*: Percentage of answers produced by a query engine when executing a query compared to the number of final query

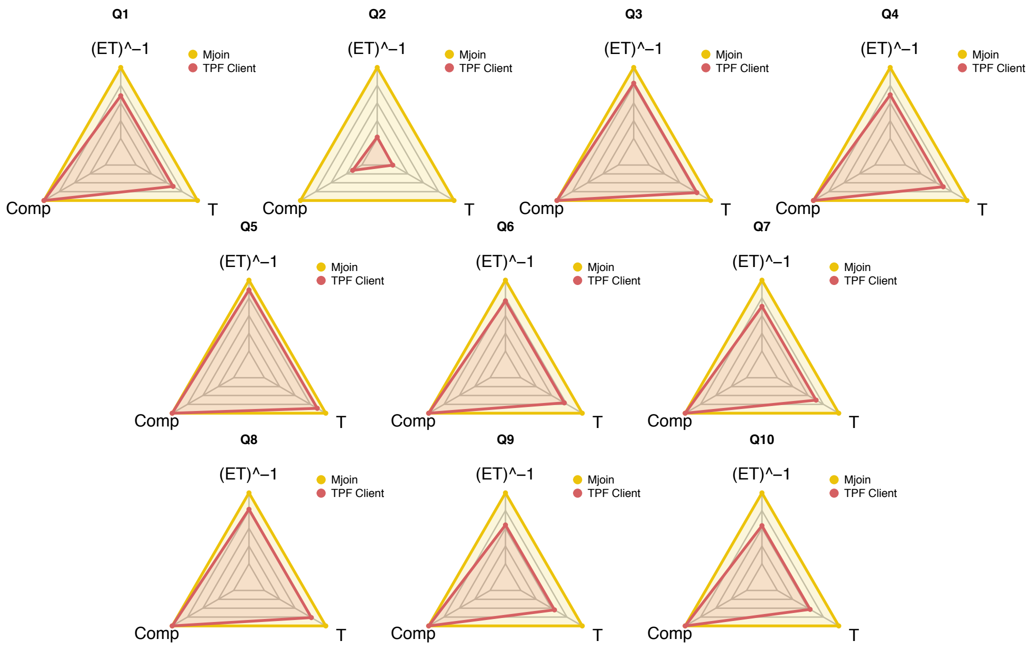


Figure 6.21: **Multidimensional comparison of binary join-only plans (TPF Client) and hybrid (Mjoin), i.e., multi-way + binary joins, plans**. In nine out of ten queries a hybrid plan outperforms a binary plan in terms of execution time and throughput at about 20% whereas completeness is retained by both approaches. In Q2 a binary plan is significantly slower and less complete. In Q5 the difference is marginal.

results. *iii*) *Throughput (T)*: Maximum number of queries that can be answered per second.
Implementation: We implemented the approach in Python 2.7.10 as an extension of the nLDE [121] query engine and SMJoin [172] multi-way join operator. Triple Pattern Fragments Server [38] was used to host HDT representation of LOD-a-lot. The experiments were executed on a Ubuntu 16.04 (64 bits) Dell PowerEdge R805 server, AMD Opteron 2.4 GHz CPU, 64 cores, 256 GB RAM.

Discussion. Fig. 6.21 reports the results obtained after executing 10 benchmark queries. Hybrid plans that combine both binary and multi-way joins tend to outperform binary-only plans at least by 20% as can be seen in nine queries except Q5 where the difference is marginal due to the high number of intermediate results to process. That is, the execution time is reduced and throughput is higher when executing high-selective queries with hybrid plans. The results support the proposed heuristics that can be considered as the first approach towards a query planner that is able to automatically analyze a query and deduce an effective hybrid plan depending on the selectivity of a triple pattern and size of a star-shaped group. Furthermore, the experiment demonstrates that SPARQL queries indeed benefit from hybrid plans and optimization techniques already known in the relational database community.

6.4.4 Evaluating MSimJoin

MSimJoin consists of two components, i.e., a similarity join operator that identifies semantically equivalent RDF molecules, SJoin, and multi-way join operator that is able to perform a join among multiple inputs simultaneously, SMJoin. We empirically evaluate the performance of each operator in Section 5.4.2 (SJoin) and Section 6.4.2 (SMJoin). Overall, the experiments show that MSimJoin similarity and multi-way components are competitive with state-of-the-art approaches providing a significant performance increase in various setups and configurations. The similarity join component, SJoin, ensures 100% query completeness in integrating syntactically different but semantically equivalent RDF molecules. The multi-way join component, SMJoin, is the fastest in delivering results of high-selective queries which constitute an important part of commonly posed queries to public SPARQL endpoints as reported by [33].

6.5 Summary

In this Chapter, we tackled the challenge of scalable semantic query processing of Big Enterprise Data that constitutes the **RQ3** of the main research problem. In order to answer the **RQ3** we proposed several query processing techniques. First, we introduced MULDER, a SPARQL query engine for federated access to SPARQL endpoints that uses RDF Molecule Templates (RDF-MTs). MULDER is significantly reducing query execution time and increasing answer completeness by using semantics.

Second, we presented SMJoin, a multi-way join operator that is tailored to support SPARQL queries and star-shaped groups. To the best of our knowledge, SMJoin is the first attempt to bring non-blocking multi-way joins to the semantic data management domain. Our future work aims at creating a query decomposition and planning approach that will efficiently combine multi-way joins with binary joins depending on the query selectivity. Moreover, SMJoin can be extended with a multi-way semantic similarity join condition to address the case when joinable tuples are semantically equivalent but encoded differently. Additionally, we showed that hybrid query plans that combine binary and multi-way join operators are able to increase query performance.

Third, we presented an approach towards a multi-way similarity join operator, MSimJoin, that tackles data duplication and redundancy by applying semantic similarity mechanisms. The operator is able to both support numerous input streams and compute similarity scores among received resources. The future work aims at providing adaptivity on the operators level and evaluating it against state-of-the-art benchmarks and RDF datasets.

With those contributions we provide empirical evidence regarding **RQ3** that Big Enterprise Data can indeed be queried in a scalable way employing advantages of semantic query processing via SPARQL on different levels, i.e., from low-level optimizations of query plans and join operators to comprehensive federated query engines. Hence, we are convinced that SPARQL and its querying technologies provide enough expressivity and performance to leverage a full potential of knowledge graphs in delivering actionable knowledge for enterprises.

CHAPTER 7

Conclusion

In this thesis, we study the research problem of transforming actionable knowledge from Big Enterprise Data. After discussing the research problem in Chapter 1, in the following chapters we provided necessary background concepts (cf. Section 2) and an overview of related work that has been done previously by the community (cf. Section 3). Then, four main chapters describe and evaluate a proposed solution. In this Chapter, we conclude the thesis by reviewing once again the stated research questions now taking into account the achieved results (cf. Section 7.1) and explore the opportunities for future work in Section 7.2.

7.1 Research Questions Analysis

The main research problem was broken down into three more specific research questions. On the way towards transforming actionable knowledge there exist four important stops that are wrapped into relevant research questions. Addressing one research question leads to a subsequent one. That is, the first research question aims at identifying an efficient actionable knowledge representation model. The second research question explores how a chosen knowledge representation model facilitates integration and fusion of knowledge from different data sources. Whereas the first two questions present more of a theoretical look on knowledge engineering, the last two concentrate on practical aspects of leveraging actionable knowledge. Hence, the third research question is dedicated to effective and efficient query mechanisms in the presence of logical and physical heterogeneity that define a range of capabilities available to the higher-level applications.

Research Question 1 (RQ1)

How can we apply the concept of knowledge graphs for representing enterprise information to become actionable knowledge?

In Chapter 4 we answer this question demonstrating that RDF as a graph-based model is capable of being a solid knowledge organization platform. We formalize a concept of an *Enterprise Knowledge Graph (EKG)*, a semantic framework based on RDF for logical representation of actionable knowledge. Being by default a semi-structured graph model on the one hand and powered by standardized logics on the other hand, an EKG exhibits better flexibility and provides valuable insights thanks to automatic machine reasoning. An EKG can serve as a comprehensive basis for creating a knowledge engineering pipeline from lower levels of data acquisition and integration to higher levels of actual implementation and application. Then, an assessment framework for comparing knowledge management technologies is devised. We compare EKGs with existing approaches and deduce that EKGs provide a wider range of opportunities for supporting enterprise data lifecycle in various aspects including human and machine interaction means as well as specific enterprise dimension such as governance and maturity. Furthermore, we illustrate how an RDF graph can be obtained from Web tables and spreadsheets which are still used by enterprises for storing and organizing their data. Having introduced EKGs we would argue that employing graph-based models for representing actionable knowledge is indeed more beneficial and rational compared to traditional relational models.

Research Question 2 (RQ2)

How can existing non-semantic data sources be lifted and semantically integrated leveraging enterprise knowledge graphs?

Chapter 5 presents a computational framework along with a set of algorithms for integrating heterogeneous RDF graphs leveraging semantics encoded in those graphs. We also demonstrate that semantic integration algorithms reduce exponential time complexity typical for baseline approaches to polynomial time complexity which, in turn, enables scalable RDF graphs integration pipelines. The outcome of the presented framework is a uniform knowledge graph that encapsulates various sources in a logically unified entity. Based on formal and empirical evidence we thus deduce about **RQ2** that semantics already encoded in enterprise data sources facilitate creation of a knowledge graph. MINTE, the implementation of the defined framework, employs a notion of RDF subgraphs and physical operators such as SJoin in order to create knowledge graphs on the fly, i.e., in an *ad-hoc* manner, as well as perform more complex integration applying specific rules which we wrapped in a concept of *fusion policies*.

Research Question 3 (RQ3)

How can we perform semantic query processing of federated enterprise data efficiently?

This research question is investigated in Chapter 6. As state-of-the-art approaches are not scalable when considering semantics, we propose new algorithms for semantic query processing and optimization that leverage semantics encoded in RDF graphs and still exhibit polynomial time complexity. MULDER is a framework for federated query processing which algorithms for query decomposition and relevant source selection practically outperform existing engines. Moreover, SMJoin introduces the first general-purpose algorithm for a multi-way join operator applicable for SPARQL queries. An implementation of this algorithm in query optimization routines is able to increase query performance in scalable setups. Furthermore, we introduce MSimJoin, an algorithm that considers both semantic similarity computations of SJoin and arbitrary input arity of SMJoin. The experiments conducted on the proposed algorithms allow us to answer **RQ3**, i.e., semantic query processing can indeed be done in a scalable way and facilitates efficient querying of federated environments typical to enterprise data storages.

As to the main challenge of *creating actionable knowledge from Big Enterprise Data*, we are able to deduce that actionable knowledge can indeed be created and integrated from Big Enterprise Data, organized into EKGs, effectively queried, and support decision making processes in various domains.

7.2 Closing Remarks and Future Work

Weaving knowledge with intelligence becomes one of the key success factors in businesses (and many other domains) of this 21st century. In this section, we discuss possible improvements, extensions and long term opportunities that open up after our work and broaden the EKG vision. As there exist a plethora of smaller but still important technical tasks we concentrate more on larger conceptual ideas.

1. *Knowledge graphs as an interpretable source for statistical AI applications.* As knowledge graphs (KGs) employ RDF they formally belong to the *symbolic reasoning* part of the Artificial Intelligence (AI) theory that puts its emphasis on logical deductions. On the other hand, the *statistical reasoning* part of AI, e.g., neural networks, received a significant traction recent years. Despite convincing results, their inner inferencing mechanisms still suffer from low interpretability due to high dimensionality. Combining rigid logical foundation with powerful statistical algorithms, e.g., in forms of embeddings of graphs or other knowledge representation formalisms, may lead to much more interpretable results how a certain neural architecture came up with a given answer. On the other hand, knowledge obtained by statistical reasoning can greatly enrich and complete contents of knowledge graphs catering for common-sense facts at a large scale usually not covered by domain specific or too high-level abstract ontologies.

2. *A methodology for knowledge graphs deployment.* As a strategy for managing enterprise knowledge, KGs should have predictable, sustainable, and reliable behavior throughout the years. Such a behavior is often supported by a standard methodology that contains best practices, recipes, instructions as well as templates for common use-cases. A standard agreed by the community would foster adaptation and implementation of recent research achievements (such as EKGs) into real-world scenarios. We also foresee a need for a KG quality assessment framework which we attempted to construct in Chapter 4, governance and maturity models for KGs that would support the deployment of the methodology on different stages of a KG lifecycle.

3. *Ethical aspects to be considered by knowledge graphs.* As any technology that leaves an academic cradle and toy examples for a worldwide use, KGs have to carefully acknowledge ethical aspects of their rich capabilities without overregulation, e.g., adhere to access control and privacy as well as evaluate possible consequences of performing certain actions. For instance, disclosing sensitive or personal information to any KG user might not be acceptable by all involved parties. The AI community slowly comes to an understanding of such needs [1], e.g., Google came up with its AI application principles [2]. Nevertheless, there is yet a long way until society fully embraces scientific and technological achievements.

[1]Web: https://medium.com/@eirinimalliaraki/toward-ethical-transparent-and-fair-ai-ml-a-critical-reading-list-d950e70a70ea

[2]Web: https://blog.google/topics/ai/ai-principles/

Bibliography

[1] T. Berners-Lee, J. Hendler and O. Lassila, *The semantic web*, Scientific american **284**.5 (2001) 28 (cit. on pp. 1, 9).

[2] W. H. Inmon, *Building the data warehouse*, John Wiley & Sons, 2005 (cit. on p. 3).

[3] A. Y. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy and S. E. Whang, *Managing Google's data lake: an overview of the Goods system.*, IEEE Data Eng. Bull. **39**.3 (2016) 5 (cit. on p. 3).

[4] R. Angles and C. Gutierrez, *Survey of graph database models*, ACM Comput. Surv. **40**.1 (2008) 1:1 (cit. on p. 4).

[5] T. Berners-Lee, R. T. Fielding and L. Masinter, *Uniform Resource Identifier (URI): Generic Syntax*, RFC **3986** (2005) 1, URL: https://doi.org/10.17487/RFC3986 (cit. on p. 9).

[6] *W3C: Extensible Markup Language (XML) 1.0*, http://www.w3.org/TR/REC-xml, URL: http://www.w3.org/TR/REC-xml (cit. on p. 9).

[7] *RDF 1.1 Concepts and Abstract Syntax*, URL: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ (cit. on pp. 9, 11, 12).

[8] T. Berners-Lee, *Linked Data - Design Issues*, URL: https://www.w3.org/DesignIssues/LinkedData.html (cit. on p. 10).

[9] A. Abele, J. P. McCrae, P. Buitelaar, A. Jentzsch and R. Cyganiak, *Linking Open Data cloud diagram 2017*, URL: http://lod-cloud.net/ (cit. on p. 11).

[10] C. Bizer, T. Heath and T. Berners-Lee, *Linked Data - The Story So Far*, Int. J. Semantic Web Inf. Syst. **5**.3 (2009) 1 (cit. on p. 11).

[11] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer and C. Bizer, *DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia*, Semantic Web **6**.2 (2015) 167 (cit. on pp. 11, 14, 29).

[12] D. Vrandecic and M. Krötzsch, *Wikidata: a free collaborative knowledgebase*, Commun. ACM **57**.10 (2014) 78 (cit. on pp. 11, 14, 29).

[13] C. Stadler, J. Lehmann, K. Höffner and S. Auer, *LinkedGeoData: A core for a web of spatial open data*, Semantic Web **3**.4 (2012) 333 (cit. on p. 11).

[14] *RDF Schema 1.1 Recommendation*, URL: https://www.w3.org/TR/2014/REC-rdf-schema-20140225/ (cit. on pp. 11, 12).

[15] J. Perez, M. Arenas and C. Gutierrez, *Semantics and complexity of SPARQL*, ACM Trans. Database Syst. **34**.3 (2009) 16:1 (cit. on pp. 12, 15–18).

[16] P. V. Biron, A. Malhotra, W. W. W. Consortium et al., *XML schema part 2: Datatypes*, 2004 (cit. on p. 12).

[17] A. Phillips and M. Davis, *Tags for identifying languages*, tech. rep., 2009, URL: `http:%20//tools.ietf.org/html/bcp47` (cit. on p. 12).

[18] T. R. Gruber, *Toward principles for the design of ontologies used for knowledge sharing?*, Int. J. Hum.-Comput. Stud. **43**.5-6 (1995) 907 (cit. on p. 13).

[19] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight and J. T. Eppig, *Gene Ontology: tool for the unification of biology*, Nature genetics **25**.1 (2000) 25 (cit. on p. 13).

[20] T. G. O. Consortium, *Expansion of the Gene Ontology knowledgebase and resources*, Nucleic Acids Research **45**.Database-Issue (2017) D331 (cit. on p. 13).

[21] K. Donnelly, *SNOMED-CT: The advanced terminology and coding system for eHealth*, Studies in health technology and informatics **121** (2006) 279 (cit. on p. 13).

[22] *OWL 2 Quick Reference*, URL: `https://www.w3.org/TR/2012/REC-owl2-quick-reference-20121211/` (cit. on p. 13).

[23] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach (3. internat. ed.)* Pearson Education, 2010 (cit. on p. 14).

[24] J. Pearl, *Probabilistic reasoning in intelligent systems - networks of plausible inference*, Morgan Kaufmann series in representation and reasoning, Morgan Kaufmann, 1989 (cit. on p. 14).

[25] A. Singhal, *Introducing the knowledge graph: things, not strings*, Official google blog (2012), URL: `https://www.blog.google/products/search/introducing-knowledge-graph-things-not/` (cit. on p. 14).

[26] F. Mahdisoltani, J. Biega and F. M. Suchanek, "YAGO3: A Knowledge Base from Multilingual Wikipedias", *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, 2015 (cit. on p. 14).

[27] R. Blanco, B. B. Cambazoglu, P. Mika and N. Torzec, "Entity Recommendations in Web Search", *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II*, 2013 33 (cit. on p. 14).

[28] H. Paulheim, *Knowledge graph refinement: A survey of approaches and evaluation methods*, Semantic Web **8**.3 (2017) 489 (cit. on pp. 14, 15).

[29] S. Vahdati, N. Arndt, S. Auer and C. Lange, "OpenResearch: Collaborative Management of Scholarly Communication Metadata", *Knowledge Engineering and Knowledge Management - 20th International Conference, EKAW 2016, Bologna, Italy, November 19-23, 2016, Proceedings*, 2016 778 (cit. on p. 15).

[30] A. Callahan, J. Cruz-Toledo, P. Ansell and M. Dumontier, "Bio2RDF Release 2: Improved Coverage, Interoperability and Provenance of Life Science Linked Data", *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*, 2013 200 (cit. on p. 15).

[31] S. Harris, A. Seaborne and E. Prud'hommeaux, *SPARQL 1.1 query language*, W3C recommendation (2013), URL: `http://www.w3.org/TR/2013/%20REC-sparql11-query-20130321/` (cit. on p. 15).

[32] M. Luczak-Roesch, B. Berendt and L. Hollink, *USEWOD 2013 Research Dataset University of Southampton*, 2013, URL: `http://dx.doi.org/10.5258/SOTON/379399` (cit. on p. 20).

[33] M. Saleem, M. I. Ali, A. Hogan, Q. Mehmood and A. N. Ngomo, "LSQ: The Linked SPARQL Queries Dataset", *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, 2015 261 (cit. on pp. 20, 118).

[34] M. Salvadores, M. Horridge, P. R. Alexander, R. W. Fergerson, M. A. Musen and N. F. Noy, "Using SPARQL to Query BioPortal Ontologies and Metadata", *The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part II*, 2012 180 (cit. on p. 20).

[35] P. Vandenbussche, J. Umbrich, L. Matteis, A. Hogan and C. B. Aranda, *SPARQLES: Monitoring public SPARQL endpoints*, Semantic Web **8**.6 (2017) 1049 (cit. on p. 20).

[36] M. Acosta, M. Vidal, F. Flöck, S. Castillo, C. B. Aranda and A. Harth, "SHEPHERD: A Shipping-Based Query Processor to Enhance SPARQL Endpoint Performance", *Proceedings of the ISWC 2014 Posters & Demonstrations Track a track within the 13th International Semantic Web Conference, ISWC 2014, Riva del Garda, Italy, October 21, 2014.* 2014 453 (cit. on p. 20).

[37] M. Acosta, M. Vidal, F. Flöck, S. Castillo and A. Harth, "PLANET: Query Plan Visualizer for Shipping Policies against Single SPARQL Endpoints", *Proceedings of the ISWC 2014 Posters & Demonstrations Track a track within the 13th International Semantic Web Conference, ISWC 2014, Riva del Garda, Italy, October 21, 2014.* 2014 189 (cit. on p. 20).

[38] R. Verborgh, M. V. Sande, O. Hartig, J. V. Herwegen, L. D. Vocht, B. D. Meester, G. Haesendonck and P. Colpaert, *Triple Pattern Fragments: A low-cost knowledge graph interface for the Web*, J. Web Sem. **37-38** (2016) 184 (cit. on pp. 20, 21, 34, 35, 103, 114, 116, 117).

[39] G. Graefe, *Query Evaluation Techniques for Large Databases*, ACM Comput. Surv. **25**.2 (1993) 73 (cit. on p. 21).

[40] T. Ibaraki and T. Kameda, *On the Optimal Nesting Order for Computing N-Relational Joins*, ACM Trans. Database Syst. **9**.3 (1984) 482 (cit. on p. 22).

[41] Y. E. Ioannidis, "Query Optimization", *The Computer Science and Engineering Handbook*, 1997 1038 (cit. on p. 22).

[42] D. Kossmann and K. Stocker,
Iterative dynamic programming: a new class of query optimization algorithms,
ACM Trans. Database Syst. **25**.1 (2000) 43 (cit. on p. 22).

[43] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie and T. G. Price,
"Access Path Selection in a Relational Database Management System",
Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, May 30 - June 1. 1979 23 (cit. on p. 22).

[44] Y. E. Ioannidis and Y. C. Kang, "Randomized Algorithms for Optimizing Large Join Queries",
Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990. 1990 312 (cit. on p. 22).

[45] A. Deshpande, Z. G. Ives and V. Raman, *Adaptive Query Processing*,
Foundations and Trends in Databases **1**.1 (2007) 1 (cit. on pp. 23, 33–35, 76, 96).

[46] M. Lenzerini, "Data Integration: A Theoretical Perspective",
Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA, 2002 233 (cit. on pp. 23–25).

[47] M. Grüninger and J. Lee, *Ontology Applications and Design - Introduction*,
Commun. ACM **45**.2 (2002) 39 (cit. on p. 24).

[48] A. Cali, D. Calvanese, G. DeGiacomo and M. Lenzerini,
"Accessing Data Integration Systems through Conceptual Schemas",
Conceptual Modeling - ER 2001, 20th International Conference on Conceptual Modeling, Yokohama, Japan, November 27-30, 2001, Proceedings, 2001 270 (cit. on p. 24).

[49] N. F. Noy, *Semantic Integration: A Survey Of Ontology-Based Approaches*,
SIGMOD Record **33**.4 (2004) 65 (cit. on p. 24).

[50] M. Lenzerini, "Ontology-Based Data Management",
Proceedings of the 6th Alberto Mendelzon International Workshop on Foundations of Data Management, Ouro Preto, Brazil, June 27-30, 2012, 2012 12 (cit. on p. 24).

[51] J. D. Ullman, "Information Integration Using Logical Views", *Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings*, 1997 19
(cit. on p. 24).

[52] A. Y. Halevy, *Answering queries using views: A survey*, VLDB J. **10**.4 (2001) 270 (cit. on p. 24).

[53] M. Fowler, *UML distilled: a brief guide to the standard object modeling language*,
Addison-Wesley Professional, 2004 (cit. on p. 29).

[54] G. Guizzardi, *Ontological foundations for structural conceptual models*,
CTIT, Centre for Telematics and Information Technology, 2005 (cit. on p. 29).

[55] O. Noran,
"UML vs. IDEF: An Ontology-Oriented Comparative Study in View of Business Modelling",
ICEIS 2004, Proceedings of the 6th International Conference on Enterprise Information Systems, Porto, Portugal, April 14-17, 2004, 2004 674 (cit. on p. 29).

[56] M. Minsky, *A framework for representing knowledge*, (1974) (cit. on p. 29).

[57] A. Horn, *On Sentences Which are True of Direct Unions of Algebras*,
J. Symb. Log. **16**.1 (1951) 14 (cit. on p. 29).

[58] R. Ahmad, *Expert Systems: Principles and Programming*, Scalable Computing: Practice and Experience **7**.4 (2006) (cit. on p. 29).

[59] C. Forgy, *Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem*, Artif. Intell. **19**.1 (1982) 17 (cit. on p. 29).

[60] M. Salatino, M. De Maio and E. Aliverti, *Mastering JBoss Drools 6*, Packt Publishing Ltd, 2016 (cit. on p. 29).

[61] C. Stadler, J. Lehmann, K. Höffner and S. Auer, *LinkedGeoData: A Core for a Web of Spatial Open Data*, Semantic Web Journal **3**.4 (2012) 333 (cit. on p. 29).

[62] X. Dong, E. Gabrilovich, G. Heitz and W. Horn, "Knowledge Vault: A Web-Scale Approach to Probabilistic Knowledge Fusion", *20th ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining*, 2014 601 (cit. on pp. 30, 47).

[63] E. Kharlamov, D. Hovland, E. Jimenez-Ruiz, D. Lanti, H. Lie, C. Pinkel, M. Rezk, M. G. Skjæveland, E. Thorstensen, G. Xiao et al., "Ontology based access to exploration data at statoil", *The Semantic Web-ISWC 2015*, Springer, 2015 (cit. on p. 30).

[64] P. Meenakshy and J. Walker, "Applying Semantic Web technologies in Product Information Management at NXP Semiconductors", *13th International Semantic Web Conference (ISWC 2014)*, 2014 (cit. on p. 30).

[65] A. Stolz, B. Rodriguez-Castro, A. Radinger and M. Hepp, "PCS2OWL: A generic approach for deriving web ontologies from product classification systems", *The Semantic Web: Trends and Challenges*, Springer, 2014 644 (cit. on p. 30).

[66] M. Osenberg, M. Langermeier and B. Bauer, "Using Semantic Web Technologies for Enterprise Architecture Analysis", *12th European Semantic Web Conference (ESWC 2015)*, Springer, 2015 668 (cit. on p. 30).

[67] B. Otto and H. Oesterle, *Corporate Data Quality. Prerequisite for Successful Business Models*, epubli, 2015 (cit. on p. 30).

[68] M. Ofner, B. Otto and H. Österle, *A Maturity Model for Enterprise Data Quality Management*, Enterprise Modelling and Information Systems Architectures **8**.2 (2013) 4 (cit. on p. 30).

[69] D. Mouromtsev, V. Vlasov, O. Parkhimovich, M. Galkin and V. Knyazev, "Development of the St. Petersburg's linked open data site using Information Workbench.", *14th FRUCT Proceedings*, 2013 77 (cit. on p. 30).

[70] M. Kolchin and F. Kozlov, "A template-based information extraction from web sites with unstable markup", *Semantic Web Evaluation Challenge*, vol. 475, 2014 89 (cit. on p. 30).

[71] A. Silva, A. Jorge and L. Torgo, *Design of an end-to-end method to extract information from tables*, International Journal of Document Analysis and Recognition (IJDAR) **8** (2006) 144 (cit. on p. 30).

[72] J. Hu, R. Kashi, D. Lopresti and G. Wilfong, *Evaluating the Performance of Table Processing Algorithms*, International Journal on Document Analysis and Recognition **4** (2002) 140 (cit. on p. 31).

[73] M. Hurst, “A constraint-based approach to table structure derivation”, *Proceedings of International Conference on Document Analysis and Recognition (ICDAR’03)*, 2003 911 (cit. on p. 31).

[74] D. Embley, C. Tao and S. Liddle, *Automating the Extraction of Data from HTML Tables with Unknown Structure*, Data and Knowledge Engineering - Special issue: ER **54** (2005) 3 (cit. on p. 31).

[75] M. A. Babyak, *What you see may not be what you get: a brief, nontechnical introduction to overfitting in regression-type models*, Psychosomatic medicine **66**.3 (2004) 411 (cit. on p. 31).

[76] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl and B. Pollak, “Towards domain- independent information extraction from web tables”, *Proceedings of the 16th WWW.* 2007 71 (cit. on p. 31).

[77] Y. Tijerino, D. Embley, D. Lonsdale, Y. Ding and G. Nagy., *Towards ontology generation from tables*, World Wide Web **8** (2005) 261 (cit. on p. 31).

[78] G. Miller and C. Fellbaum, *Wordnet: An electronic lexical database*, 1998 (cit. on p. 31).

[79] P. M. V. Crescenzi G. Mecca, “RoadRunner: Towards automatic data extraction from large websites”, *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB’01)*, 2001 109 (cit. on p. 31).

[80] C. A. Knoblock, P. A. Szekely, J. L. Ambite, A. Goel, S. Gupta, K. Lerman, M. Muslea, M. Taheriyan and P. Mallick, “Semi-automatically Mapping Structured Sources into the Semantic Web”, *ESWC - 9th Extended Semantic Web Conference, Heraklion, Crete, Greece*, 2012 375 (cit. on pp. 31, 32).

[81] S. Choudhury, K. Agarwal, S. Purohit, B. Zhang, M. Pirrung, W. Smith and M. Thomas, *NOUS: Construction and Querying of Dynamic Knowledge Graphs*, arXiv preprint arXiv:1606.02314 (2016) (cit. on p. 32).

[82] T. Palomares, Y. Ahres, J. Kangaspunta and C. Re, “Wikipedia Knowledge Graph with DeepDive”, *10th International AAAI Conference on Web and Social Media*, 2016 (cit. on p. 32).

[83] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr and T. M. Mitchell, “Toward an Architecture for Never-Ending Language Learning.”, *AAAI*, vol. 5, 2010 3 (cit. on p. 32).

[84] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun and W. Zhang, “Knowledge vault: A web-scale approach to probabilistic knowledge fusion”, *SIGKDD*, ACM, 2014 601 (cit. on p. 32).

[85] M. W. Chekol, G. Pirro, J. Schoenfisch and H. Stuckenschmidt, “Marrying Uncertainty and Time in Knowledge Graphs”, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.* 2017 88 (cit. on p. 32).

[86] A. Schultz, A. Matteini, R. Isele, P. N. Mendes, C. Bizer and C. Becker, “Ldif-a framework for large-scale linked data integration”, *21st Int. World Wide Web Conference (WWW 2012), Developers Track*, 2012 (cit. on p. 32).

[87] J. Michelfeit, T. Knap and M. Nečasky, *Linked Data Integration with Conflicts*, arXiv preprint arXiv:1410.7990 (2014) (cit. on p. 32).

[88] A.-C. Ngonga Ngomo and S. Auer, "LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data", *IJCAI*, 2011 (cit. on p. 32).

[89] V. Fionda and G. Pirro, *Explaining and Querying Knowledge Graphs by Relatedness*, PVLDB **10**.12 (2017) 1913 (cit. on p. 32).

[90] R. Isele and C. Bizer, *Active learning of expressive linkage rules using genetic programming*, J. Web Sem. **23** (2013) 2 (cit. on p. 32).

[91] P. N. Mendes, H. Mühleisen and C. Bizer, "Sieve: linked data quality assessment and fusion", *Joint EDBT/ICDT Workshops, Berlin, Germany*, 2012 116 (cit. on p. 32).

[92] J. Michelfeit and T. Knap, "Linked Data Fusion in ODCleanStore", *ISWC - International Semantic Web Conference, Posters & Demonstrations Track, Boston, USA*, 2012 (cit. on p. 32).

[93] T. Knap, M. Kukhar, B. Machac, P. Skoda, J. Tomes and J. Vojt, "UnifiedViews: An ETL Framework for Sustainable RDF Data Processing", *ESWC - The Semantic Web, Satellite Events, Anissaras, Crete, Greece*, 2014 379 (cit. on p. 32).

[94] J. Michelfeit, T. Knap and M. Necasky, *Linked Data Integration with Conflicts*, CoRR **abs/1410.7990** (2014), URL: `http://arxiv.org/abs/1410.7990` (cit. on p. 32).

[95] M. Acosta, M. Vidal, T. Lampo, J. Castillo and E. Ruckhaus, "ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints", *ISWC - 10th International Semantic Web Conference, Bonn, Germany*, 2011 18 (cit. on pp. 33–35, 93, 96, 97, 110).

[96] A. Schwarte, P. Haase, K. Hose, R. Schenkel and M. Schmidt, "FedX: Optimization Techniques for Federated Query Processing on Linked Data", *ISWC - 10th International Semantic Web Conference, Bonn, Germany*, 2011 601 (cit. on pp. 33–35, 93).

[97] O. Görlitz and S. Staab, "SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions", *COLD - Second International Workshop on Consuming Linked Data, Bonn, Germany*, 2011 13 (cit. on p. 33).

[98] G. Ladwig and T. Tran, "SIHJoin: Querying Remote and Local Linked Data", *ESWC - 8th Extended Semantic Web Conference, Heraklion, Crete, Greece*, 2011 139 (cit. on p. 33).

[99] T. Urhan and M. J. Franklin, *XJoin: A Reactively-Scheduled Pipelined Join Operator*, IEEE Data Eng. Bull. **23**.2 (2000) 27 (cit. on pp. 33, 76).

[100] S. Wandelt, D. Deng, S. Gerdjikov, S. Mishra, P. Mitankin, M. Patil, E. Siragusa, A. Tiskin, W. Wang, J. Wang and U. Leser, *State-of-the-art in string similarity search and join*, SIGMOD Record **43**.1 (2014) 64 (cit. on pp. 33, 76).

[101] J. Feng, J. Wang and G. Li, *Trie-join: a trie-based method for efficient string similarity joins*, VLDB J. **21**.4 (2012) 437 (cit. on pp. 33, 76).

[102] G. Li, D. Deng, J. Wang and J. Feng, *PASS-JOIN: A Partition-based Method for Similarity Joins*, PVLDB **5**.3 (2011) 253 (cit. on pp. 33, 76).

[103] W. Mann, N. Augsten and P. Bouros, *An Empirical Evaluation of Set Similarity Join Techniques*, PVLDB **9**.9 (2016) 636 (cit. on pp. 33, 76).

[104] L. A. Ribeiro, A. Cuzzocrea, K. A. A. Bezerra and B. H. B. do Nascimento, "Incorporating Clustering into Set Similarity Join Algorithms: The SjClust Framework", *DEXA 2016, Porto, Portugal*, 2016 185 (cit. on pp. 33, 76).

[105] Y. Wang, H. Wang, J. Li and H. Gao, *Efficient graph similarity join for information integration on graphs*, Frontiers of Computer Science **10**.2 (2016) 317 (cit. on p. 33).

[106] Z. Shang, Y. Liu, G. Li and J. Feng, *K-Join: Knowledge-Aware Similarity Join*, IEEE Trans. Knowl. Data Eng. **28**.12 (2016) 3293 (cit. on p. 33).

[107] H. Zhu, X. Meng and G. Kollios, *NED: An Inter-Graph Node Metric Based On Edit Distance*, PVLDB **10**.6 (2017) 697 (cit. on p. 33).

[108] I. T. Ribon, M. Vidal, B. Kämpgen and Y. Sure-Vetter, "GADES: A Graph-based Semantic Similarity Measure", *SEMANTICS - 12th International Conference on Semantic Systems, Leipzig, Germany*, 2016 101 (cit. on pp. 33, 79, 86, 107).

[109] M. Saleem, Y. Khan, A. Hasnain, I. Ermilov and A. N. Ngomo, *A fine-grained evaluation of SPARQL endpoint federation systems*, Semantic Web **7**.5 (2015) (cit. on p. 34).

[110] C. Basca and A. Bernstein, *Querying a messy web of data with Avalanche*, J. Web Semantics **26** (2014) 1 (cit. on pp. 34, 35).

[111] K. Alexander and M. Hausenblas, "Describing Linked Datasets-On the Design and Usage of voiD, the 'Vocabulary of Interlinked Datasets'", *LDOW*, 2009 (cit. on p. 34).

[112] M. Saleem and A. N. Ngomo, "HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation", *ESWC*, 2014 176, URL: `http://dx.doi.org/10.1007/978-3-319-07443-6_13` (cit. on pp. 34, 93).

[113] G. Montoya, H. Skaf-Molli, P. Molli and M. Vidal, "Federated SPARQL Queries Processing with Replicated Fragments", *ISWC 2015*, 2015 36 (cit. on p. 34).

[114] M. Wylot and P. Cudre-Mauroux, *DiploCloud: Efficient and Scalable Management of RDF Data in the Cloud*, IEEE Trans. Knowl. Data Eng. **28**.3 (2016) 659 (cit. on p. 34).

[115] M. Vidal, S. Castillo, M. Acosta, G. Montoya and G. Palma, *On the Selection of SPARQL Endpoints to Efficiently Execute Federated SPARQL Queries*, Trans. Large-Scale Data- and Knowledge-Centered Systems **25** (2016) 109 (cit. on pp. 34, 95, 105, 110).

[116] G. Montoya, H. Skaf-Molli, P. Molli and M. Vidal, *Decomposing federated queries in presence of replicated fragments*, J. Web Sem. **42** (2017) 1 (cit. on p. 34).

[117] H. Garcia-Molina, J. D. Ullman and J. Widom, *Database System Implementation*, Prentice-Hall, 2000 (cit. on p. 34).

[118] T. Urhan and M. J. Franklin, *XJoin: A Reactively-Scheduled Pipelined Join Operator*, IEEE Data Eng. Bull. **23**.2 (2000) 27 (cit. on pp. 34, 96).

[119] D. Florescu, A. Y. Levy, I. Manolescu and D. Suciu, "Query Optimization in the Presence of Limited Access Patterns", *SIGMOD 1999, USA*. 1999 311 (cit. on p. 34).

[120] G. Ladwig and T. Tran, "SIHJoin: Querying Remote and Local Linked Data", *8th ESWC, Greece*, 2011 139 (cit. on pp. 35, 96).

[121] M. Acosta and M. Vidal, "Networks of Linked Data Eddies: An Adaptive Web Query Processing Engine for RDF Data", *14th ISWC, USA*, 2015 111 (cit. on pp. 35, 96, 103, 114, 117).

[122] S. Viglas, J. F. Naughton and J. Burger, "Maximizing the Output Rate of Multi-Way Join Queries over Streaming Information Sources", *29th VLDB, Germany*, 2003 285 (cit. on pp. 35, 96, 97).

[123] H. Q. Ngo, E. Porat, C. Re and A. Rudra, "Worst-case optimal join algorithms: [extended abstract]", *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART 2012*, 2012 37 (cit. on p. 35).

[124] A. Atserias, M. Grohe and D. Marx, "Size Bounds and Query Plans for Relational Joins", *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, 2008 739 (cit. on p. 35).

[125] T. L. Veldhuizen, "Triejoin: A Simple, Worst-Case Optimal Join Algorithm", *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014.* 2014 96 (cit. on p. 35).

[126] B. Gedik, K. Wu, P. S. Yu and L. Liu, *GrubJoin: An Adaptive, Multi-Way, Windowed Stream Join with Time Correlation-Aware CPU Load Shedding*, IEEE Trans. Knowl. Data Eng. **19**.10 (2007) 1363 (cit. on pp. 35, 96).

[127] X. Zhang, L. Chen and M. Wang, *Efficient Multi-way Theta-Join Processing Using MapReduce*, PVLDB **5**.11 (2012) 1184 (cit. on pp. 35, 96).

[128] J. Ahn, D. Im and H. Kim, *SigMR: MapReduce-based SPARQL query processing by signature encoding and multi-way join*, The Journal of Supercomputing **71**.10 (2015) 3695 (cit. on p. 35).

[129] D. L. Phuoc, H. N. M. Quoc, C. L. Van and M. Hauswirth, "Elastic and Scalable Processing of Linked Stream Data in the Cloud", *12th ISWC, Australia, 2013*, 2013 280 (cit. on p. 35).

[130] O. Erling and I. Mikhailov, "Virtuoso: RDF Support in a Native RDBMS", *Semantic Web Information Management - A Model-Based Perspective*, 2009 501 (cit. on p. 35).

[131] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev and R. Velkov, *OWLIM: A family of scalable semantic repositories*, Semantic Web **2**.1 (2011) 33 (cit. on p. 35).

[132] X. Ren and O. Cure, "Strider: A Hybrid Adaptive Distributed RDF Stream Processing Engine", *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I*, 2017 559 (cit. on p. 35).

[133] M. Galkin, S. Auer, H. L. Kim and S. Scerri,
"Integration Strategies for Enterprise Knowledge Graphs", *Tenth IEEE International Conference on Semantic Computing, ICSC 2016, Laguna Hills, CA, USA, February 4-6, 2016*, 2016 242 (cit. on p. 38).

[134] M. Galkin, S. Auer and S. Scerri,
"Enterprise Knowledge Graphs: A Backbone of Linked Enterprise Data", *2016 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2016, Omaha, NE, USA, October 13-16, 2016*, 2016 497 (cit. on p. 38).

[135] M. Galkin, S. Auer, M. Vidal and S. Scerri,
"Enterprise Knowledge Graphs: A Semantic Approach for Knowledge Management in the Next Generation of Enterprise Information Systems",
ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems, Volume 2, Porto, Portugal, April 26-29, 2017, 2017 88 (cit. on p. 38).

[136] D. Hislop, *Knowledge management in organizations: A critical introduction*,
Oxford University Press, 2013 (cit. on p. 38).

[137] Q. Miao, Y. Meng and B. Zhang,
"Chinese enterprise knowledge graph construction based on Linked Data",
Semantic Computing (ICSC), 2015 IEEE International Conference on, IEEE, 2015 153 (cit. on p. 38).

[138] M. Nickel, K. Murphy, V. Tresp and E. Gabrilovich,
A review of relational machine learning for knowledge graphs,
Proceedings of the IEEE **104**.1 (2016) 11 (cit. on p. 38).

[139] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak and S. Hellmann,
DBpedia-A crystallization point for the Web of Data,
Web Semantics: science, services and agents on the world wide web **7**.3 (2009) 154 (cit. on p. 38).

[140] D. Romero and F. B. Vernadat,
Future perspectives on next generation enterprise information systems,
Computers in Industry **79** (2016) 1 (cit. on p. 39).

[141] J. D. Ullman, "Information integration using logical views",
International Conference on Database Theory, Springer, 1997 19 (cit. on p. 42).

[142] P. Frischmuth, S. Auer, S. Tramp, J. Unbehauen, K. Holzweissig and C. Marquardt,
"Towards Linked Data based Enterprise Information Integration",
Workshop on Semantic Web Enterprise Adoption and Best Practice (ISWC 2013), 2013.
Ed. by S. Coppens, K. Hammar and M. Knuth, CEUR-WS.org, 2013 (cit. on p. 43).

[143] T. Pellegrini, H. Sack and S. Auer, eds., *Linked Enterprise Data*, X.media.press, Springer, 2014 (cit. on p. 43).

[144] O. Deshpande, D. S. Lamba and et al.,
"Building, maintaining, and using knowledge bases: A report from the trenches",
2013 ACM SIGMOD International Conference on Management of Data, ACM, 2013 1209 (cit. on p. 47).

[145] Y. Chen and D. Z. Wang, “Knowledge expansion over probabilistic knowledge bases”, *2014 ACM SIGMOD international conference on Management of data*, ACM, 2014 649 (cit. on p. 47).

[146] M. Niepert and P. Domingos, “Tractable Probabilistic Knowledge Bases: Wikipedia and Beyond”, *Workshops at the 28th AAAI Conference on Artificial Intelligence*, 2014 (cit. on p. 47).

[147] J. Huber, C. Meilicke and H. Stuckenschmidt, “Applying Markov Logic for Debugging Probabilistic Temporal Knowledge Bases”, *4th Workshop on Automated Knowledge Base Construction (AKBC)*, 2014 (cit. on p. 47).

[148] X. Li and B. Liu, *Hybrid logic and uncertain logic*, Journal of Uncertain Systems **3**.2 (2009) 83 (cit. on p. 47).

[149] K. M. Endris, S. Faisal, F. Orlandi, S. Auer and S. Scerri, “Interest-based RDF Update Propagation”, *14th International Semantic Web Conference (ISWC 2015)*, 2015 (cit. on p. 48).

[150] J. Weston, A. Bordes, S. Chopra and T. Mikolov, *Towards ai-complete question answering: A set of prerequisite toy tasks*, arXiv preprint arXiv:1502.05698 (2015) (cit. on p. 50).

[151] S. Shekarpour, D. Lukovnikov, A. J. Kumar, K. Endris, K. Singh, H. Thakkar and C. Lange, *Question Answering on Linked Data: Challenges and Future Directions*, arXiv preprint arXiv:1601.03541 (2016) (cit. on p. 50).

[152] Delphi, *Information Intelligence: Content Classification and the Enterprise Taxonomy Practice*, Whitepaper, 2004 (cit. on p. 50).

[153] A. Wroblewska, P. Kaplanski, P. Zarzycki and I. Lugowska, “Semantic rules representation in controlled natural language in FluentEditor”, *Human System Interaction (HSI), 2013 The 6th International Conference on*, IEEE, 2013 90 (cit. on p. 52).

[154] A.-D. Mezaour, B. Van Nuffelen and C. Blaschke, “Building Enterprise Ready Applications Using Linked Open Data”, *Linked Open Data–Creating Knowledge Out of Interlinked Data*, Springer, 2014 155 (cit. on p. 52).

[155] T. Schandl and A. Blumauer, “PoolParty: SKOS thesaurus management utilizing linked data”, *The Semantic Web: Research and Applications*, Springer, 2010 (cit. on p. 52).

[156] M. Rospocher, M. van Erp, P. Vossen, A. Fokkens, I. Aldabe, G. Rigau, A. Soroa, T. Ploeger and T. Bogaard, *Building event-centric knowledge graphs from news*, Web Semantics: Science, Services and Agents on the World Wide Web (2016) (cit. on p. 53).

[157] M. Galkin, D. Mouromtsev and S. Auer, “Identifying Web Tables: Supporting a Neglected Type of Content on the Web”, *Knowledge Engineering and Semantic Web - 6th International Conference, KESW 2015, Moscow, Russia, September 30 - October 2, 2015, Proceedings*, 2015 48 (cit. on p. 66).

[158] D. Collarana, M. Galkin, I. T. Ribon, C. Lange, M. Vidal and S. Auer, “Semantic Data Integration for Knowledge Graph Construction at Query Time”, *11th IEEE International Conference on Semantic Computing, ICSC*, 2017 109 (cit. on p. 66).

[159] D. Collarana, M. Galkin, I. T. Ribon, M. Vidal, C. Lange and S. Auer, "MINTE: semantically integrating RDF graphs", *Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics, WIMS 2017*, 2017 22:1 (cit. on pp. 66, 73).

[160] M. Galkin, D. Collarana, I. T. Ribon, M. Vidal and S. Auer, "SJoin: A Semantic Join Operator to Integrate Heterogeneous RDF Graphs", *Database and Expert Systems Applications - 28th International Conference, DEXA 2017, Proceedings, Part I*, 2017 206 (cit. on p. 66).

[161] M. Cafarella, E. Wu, A. Halevy, Y. Zhang and D. Wang, "WebTables: exploring the power of tables on the web", *VLDB 2008*, 2008 (cit. on pp. 67, 82).

[162] E. Crestan and P. Pantel, "Web-scale table census and classification", *WSDM 2011 Proceedings of the fourth ACM international conference on Web search and data mining*, ed. by ACM, 2011 545 (cit. on p. 67).

[163] I. Ermilov, S. Auer and C. Stadler, "User-driven semantic mapping of tabular data", *Proceedings of the 9th International Conference on Semantic Systems*, ACM, 2013 105 (cit. on p. 68).

[164] T. Mitchell, *Machine Learning*, McGraw-Hill Science/Engineering/Math, 1997 (cit. on pp. 69, 82).

[165] J. H. Y. Wang, "A machine learning based approach for table detection on the web", *Proceedings of the 11th WWW*, 2002 242 (cit. on p. 70).

[166] J. D. Fernandez, A. Llaves and O. Corcho, "Efficient RDF Interchange (ERI) Format for RDF Data Streams", *ISWC - 13th International Semantic Web Conference, Riva del Garda, Italy*, 2014 244 (cit. on pp. 73, 76).

[167] J. Munkres, *Algorithms for the assignment and transportation problems*, Journal of the society for industrial and applied mathematics **5**.1 (1957) 32 (cit. on p. 81).

[168] L. Rokach and O. Maimon, *Data mining with decision trees: theory and applications*, World Scientific Pub Co Inc, 2008 (cit. on p. 82).

[169] J. Tou and R. Gonzalez, *Pattern Recognition Principles*, ed. by R. Kalaba, University of Southern California, 1974 (cit. on p. 82).

[170] D. Collarana, M. Galkin, C. Lange, I. Grangel-Gonzalez, M. Vidal and S. Auer, "FuhSen: A Federated Hybrid Search Engine for Building a Knowledge Graph On-Demand (Short Paper)", *OTM - Confederated International Conferences: CoopIS, C&TC, and ODBASE, Rhodes, Greece, 2016*, 2016 752 (cit. on p. 86).

[171] K. M. Endris, M. Galkin, I. Lytra, M. N. Mami, M. Vidal and S. Auer, *Querying Interlinked Data by Bridging RDF Molecule Templates*, Accepted for publication in the LNCS Transactions on Large-Scale Data- and Knowledge-Centered Systems Journal (Transactions LDKS) (2018) (cit. on pp. 92, 93).

[172] M. Galkin, K. M. Endris, M. Acosta, D. Collarana, M.-E. Vidal and S. Auer, "SMJoin: A Multi-way Join Operator for SPARQL queries", *13th International Conference on Semantic Systems, SEMANTICS, Amsterdam, 2017*, 2017 104 (cit. on pp. 92, 117).

[173] M. Galkin, M. Vidal and S. Auer, “Towards a Multi-way Similarity Join Operator”, *New Trends in Databases and Information Systems - ADBIS 2017 Short Papers and Workshops, AMSD, BigNovelTI, DAS, SW4CH, DC, Nicosia, Cyprus, September 24-27, 2017, Proceedings*, 2017 267 (cit. on p. 92).

[174] O. Görlitz and S. Staab, “SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions”, *Proceedings of the Second International Workshop on Consuming Linked Data (COLD2011), Bonn, Germany, October 23, 2011*, 2011 (cit. on p. 93).

[175] J. D. Fernandez, W. Beek, M. A. Martinez-Prieto and M. Arias, “LOD-a-lot - A Queryable Dump of the LOD Cloud”, *The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II*, 2017 75 (cit. on pp. 103, 116).

[176] C. Buil-Aranda, M. Arenas, O. Corcho and A. Polleres, *Federating Queries in SPARQL1.1: Syntax, Semantics and Evaluation*, Web Semantics: Science, Services and Agents on the World Wide Web **18**.0 (2013) (cit. on p. 105).

[177] C. Morales, D. Collarana, M. Vidal and S. Auer, “MateTee: A Semantic Similarity Metric Based on Translation Embeddings for Knowledge Graphs”, *Web Engineering - 17th International Conference, ICWE 2017, Italy, 2017, Proceedings*, 2017 246 (cit. on p. 108).

[178] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on scientific Computing **20**.1 (1998) (cit. on p. 110).

[179] G. Palma, M. Vidal and L. Raschid, “Drug-Target Interaction Prediction Using Semantic Similarity and Edge Partitioning”, *13th ISWC*, 2014 (cit. on p. 110).

[180] J. D. Fernandez, M. A. Martinez-Prieto, C. Gutierrez, A. Polleres and M. Arias, *Binary RDF representation for publication and exchange (HDT)*, J. Web Sem. **19** (2013) 22 (cit. on p. 116).

Appendix

APPENDIX A

List of Publications

- *Conference Papers*
  1. **Mikhail Galkin**, Dmitry Mouromtsev, Sören Auer. *Identifying Web Tables: Supporting a Neglected Type of Content on the Web*. In Proceedings of the 6th International Conference on Knowledge Engineering and Semantic Web (KESW) 2015, 48–62, Springer, **2nd Best Student Paper Award**;
  2. Klaudia Thellmann, **Michael Galkin**, Fabrizio Orlandi, Sören Auer. *LinkDaViz - Automatic Binding of Linked Data to Visualizations*. In Proceedings of the 14th International Semantic Web Conference (ISWC) 2015, 147–162, Springer;
  3. **Mikhail Galkin**, Sören Auer, Hak Lae Kim, Simon Scerri. *Integration Strategies for Enterprise Knowledge Graphs.* In Proceedings of the 10th IEEE International Conference on Semantic Computing (ICSC) 2016, 242–245, IEEE;
  4. Diego Collarana, **Mikhail Galkin**, Christoph Lange, Irlán Grangel-González, Maria-Esther Vidal, Sören Auer. *FuhSen: A Federated Hybrid Search Engine for Building a Knowledge Graph On-Demand.* In Proceedings of the On the Move to Meaningful Internet Systems OTM 2016 Conferences - Confederated International Conferences CoopIS, CTC, and ODBASE (ODBASE) 2016, 752–761, Springer;
  5. **Mikhail Galkin**, Sören Auer, Simon Scerri. *Enterprise Knowledge Graphs: A Backbone of Linked Enterprise Data.* In Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence (WI) 2016, 497–502, ACM;
  6. Niklas Petersen, **Michael Galkin**, Christoph Lange, Steffen Lohmann, Sören Auer. *Monitoring and Automating Factories Using Semantic Models.* In Proceedings of the 6th Joint International Conference on Semantic Technology (JIST) 2016, 315–330, Springer;
  7. Diego Collarana, **Mikhail Galkin**, Ignacio Traverso-Ribón, Christoph Lange, Maria-Esther Vidal, Sören Auer. *Semantic Data Integration for Knowledge Graph Construction at Query Time.* In Proceedings of the 11th IEEE International Conference on Semantic Computing (ICSC) 2017, 109–116, IEEE;
  8. **Mikhail Galkin**, Sören Auer, Maria-Esther Vidal, Simon Scerri. *Enterprise Knowledge Graphs: A Semantic Approach for Knowledge Management in the Next Generation of Enterprise Information Systems.* In Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS) 2017, Volume 2, 88–98, Scitepress, **Best Student Paper Nominee**;

9. Diego Collarana, **Mikhail Galkin**, Ignacio Traverso-Ribón, Christoph Lange, Maria-Esther Vidal, Sören Auer. *MINTE: semantically integrating RDF graphs.* In Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics (WIMS) 2017, 22:1–22:11, ACM;

10. **Mikhail Galkin**, Diego Collarana, Ignacio Traverso-Ribón, Christoph Lange, Maria-Esther Vidal, Sören Auer. *SJoin: A Semantic Join Operator to Integrate Heterogeneous RDF Graphs.* In Proceedings of the 28th International Conference of Database and Expert Systems Applications (DEXA) 2017, 206–221, Springer;

11. Kemele M. Endris, **Mikhail Galkin**, Ioanna Lytra, Mohamed Nadjib Mami, Maria-Esther Vidal, Sören Auer. *MULDER: Querying the Linked Data Web by Bridging RDF Molecule Templates.* In Proceedings of the 28th International Conference of Database and Expert Systems Applications (DEXA) 2017, 3–18, Springer, **Best Paper Award**;

12. **Mikhail Galkin**, Kemele M. Endris, Maribel Acosta, Diego Collarana, Maria-Esther Vidal, Sören Auer. *SMJoin: A Multi-way Join Operator for SPARQL Queries.* In Proceedings of the 13th International Conference on Semantic Systems (SEMANTiCS) 2017, 104–111, Springer;

13. Omar Al-Safi, Christian Mader, Ioanna Lytra, **Mikhail Galkin**, Kemele M. Endris, Maria-Esther Vidal and Sören Auer. *Shipping Knowledge Graph Management Capabilities to Data Providers and Consumers.* In Proceedings of the 12th IEEE International Conference on Semantic Computing (ICSC) 2018, IEEE;

14. Diego Collarana, **Mikhail Galkin**, Simon Scerri, Christoph Lange, Sören Auer, Maria-Esther Vidal,. *Synthesizing Knowledge Graphs from Web Sources with the MINTE$^+$ Framework.* Accepted for publication at the 17th International Semantic Web Conference (ISWC) 2018.

- *Workshops and Demos*

15. Dmitry Mouromtsev, Dmitry Pavlov, Yury Emelyanov, Alexey Morozov, Daniil Razdyakonov, **Mikhail Galkin**. *The Simple Web-based Tool for Visualization and Sharing of Semantic Data and Ontologies.* In Posters and Demo Track at the 14th International Semantic Web Conference (ISWC) 2015, CEUR-WS Vol 1486;

16. **Mikhail Galkin**, Maria-Esther Vidal, Sören Auer. *Towards a Multi-way Similarity Join Operator.* In Proceedings of the 21st European Conference on Advances in Databases and Information Systems (ADBIS) 2017 Short Papers and Workshops, Springer;

17. **Mikhail Galkin**, Maria-Esther Vidal. *BatWAn: A Binary and Multi-Way Query Plan Analyzer.* In Posters and Demo Track at the 16th International Semantic Web Conference (ISWC) 2017, CEUR-WS Vol 1963.

18. **Mikhail Galkin**, Diego Collarana, Maria-Esther Vidal,. *Synthesizing Data Scientist Job Offers with MINTE$^+$.* Accepted for publication at the 17th International Semantic Web Conference (ISWC) 2018 Posters and Demo Track.

- *Journal Articles*

19. Kemele M. Endris, **Mikhail Galkin**, Ioanna Lytra, Mohamed Nadjib Mami, Maria-Esther Vidal, Sören Auer. *MULDER: Querying the Linked Data Web by Bridging RDF Molecule Templates.* Accepted for publication in the LNCS Transactions on Large-Scale Data- and Knowledge-Centered Systems Journal (Transactions LDKS), 2018;

List of Figures

List of Tables