Scalable Extraction and Adoption of Shapes for Improving Data Quality and Query Processing in Knowledge Graphs

Rabbani, Kashif

DOI (link to publication from Publisher):
10.54337/aau740075028

Publication date:
2024

Document Version
Publisher's PDF, also known as Version of record

SCALABLE EXTRACTION AND ADOPTION OF SHAPES FOR IMPROVING DATA QUALITY AND QUERY PROCESSING IN KNOWLEDGE GRAPHS

BY
KASHIF RABBANI

PhD Thesis 2024

AALBORG UNIVERSITY
DENMARK

Scalable Extraction and Adoption of Shapes for Improving Data Quality and Query Processing in Knowledge Graphs

Ph.D. Dissertation
Kashif Rabbani

Dissertation submitted May, 2024

Submitted: May 2024

Main Supervisor: Professor Katja Hose
Aalborg University, Denmark
TU Wien, Austria

Co-supervisor: Associate Professor Matteo Lissandrini
Aalborg University, Denmark
University of Verona, Italy

Assessment: Professor Jiri Srba (chair)
Aalborg University, Denmark

Associate Professor Semih Salihoğlu
University of Waterloo, Canada

Associate Professor Paolo Papotti
EURECOM, France

PhD Series: Technical Faculty of IT and Design, Aalborg University

Department: Department of Computer Science

ISSN: 2446-1628
ISBN: 978-87-94563-39-0

Published by:
Aalborg University Open Publishing
Kroghstræde 1-3
DK – 9220 Aalborg Øst
aauopen@aau.dk

Abstract

Over the past two decades, the Semantic Web has evolved significantly, providing access to a vast and interconnected Web of Data, often referred to as Linked Open Data. Within this landscape, Knowledge Graphs (KGs) have emerged as powerful embodiments of interconnected data and semantic relationships, leveraging Semantic Web standards. KGs play a pivotal role in various domains, offering practical realizations of the vision of the Web of Data. However, as KGs continue to evolve, several key challenges have become prominent: the quality of data within KGs, efficient data access mechanisms, and interoperability between different KG data models. This thesis investigates and addresses these challenges, proposing techniques to improve data quality, enable efficient data access, and enhance interoperability in KGs.

To enhance data quality within KGs we investigate the generation and adoption of constraints on KGs in the form of validating shapes. We conduct an extensive community survey to gain insights into the current state of the art. The survey findings underline the necessity of developing semi-automatic methods for generating validating shapes, thus addressing the need for tools for assessing and enhancing data quality in KGs. Building on these results, we introduce Quality Shapes Extraction (QSE), an efficient and scalable approach designed to extract quality shapes from very large KGs. QSE significantly reduces extraction time and filters out invalid and spurious shapes, contributing to enhancing data quality in KGs. Building upon QSE, we extend the methodology to improve the quality of data in KGs, considering factors such as completeness and accuracy. To do so, we propose SHACTOR, a system tailored for extracting and analyzing shape constraints and utilizing them to detect and correct erroneous and spurious data in the KGs.

To address the data access challenge, we leverage validating shapes to optimize SPARQL query processing over KGs. We propose "shapes statistics" for cardinality estimation during query planning, resulting in efficient query optimization. We demonstrate the effectiveness of this approach on both synthetic and real-world datasets, demonstrating its potential to improve query performance in large KGs.

Furthermore, we introduce S3PG, a novel approach aimed at enhancing data interoperability of two popular data models for KGs: RDF and property graphs. S3PG ensures lossless and monotonic transformation from RDF to property graph, contributing to improving data interoperability in KGs by enabling seamless data exchange between different KG data models.

In summary, this thesis contributes to advancing the state of the art in data management for KG research by proposing novel methodologies and techniques. It fosters the practical utilization of KGs in various domains, including but not limited to information retrieval, data integration, and knowledge discovery. The methodologies and tools developed and published as research papers in the context of this thesis make valuable contributions to ongoing efforts to enhance the quality, accessibility, and interoperability of KGs, thereby paving the way for their extensive adoption and profound impact across various industries, particularly in the realm of artificial intelligence.

Resumé

I løbet af de sidste to årtier er det Semantiske Web blevet udviklet væsentligt og har skabt adgang til et stort og sammenkoblet Web af Data, ofte omtalt som Linked Open Data. Inden for dette er vidensgrafer opstået som kraftfulde inkarnationer af sammenkoblede data og semantiske relationer, der udnytter standarder fra det Semantiske Web. Vidensgrafer spiller en afgørende rolle i forskellige domæner og hvor visionen om et Web af Data bliver realiseret. Ikke desto mindre, mens vidensgrafer fortsætter med at proliferere, er flere nøgleudfordringer blevet prominente: kvaliteten af data inden for vidensgrafer, effektive dataadgangsmekanismer og interoperabilitet mellem forskellige datamodeller. Denne afhandling undersøger og løser disse udfordringer og foreslår teknikker til at forbedre datakvaliteten, muliggøre effektiv dataadgang og forbedre interoperabiliteten i vidensgrafer.

For at forbedre datakvaliteten inden for vidensgrafer undersøger vi generering og adoption af begrænsninger på vidensgrafer i form af validerende former. Vi gennemfører en omfattende fællesskabsundersøgelse for at få indblik i state-of-the-art. Undersøgelsens resultater understreger nødvendigheden af at udvikle semi-automatiske metoder til at generere validerende former, hvilket adresserer behovet for at forbedre datakvaliteten i vidensgrafer. På baggrund af disse resultater introducerer vi Quality Shapes Extraction (QSE), en effektiv og skalerbar tilgang designet til at udtrække kvalitetsformer fra meget store vidensgrafer. QSE reducerer signifikant udtrækningsprocessens tid og filtrerer ugyldige og forkerte former, hvilket bidrager til at forbedre datakvaliteten i vidensgrafer. Ved at bygge på QSE udvider vi metodologien for yderligere at forbedre kvaliteten af data i vidensgrafer med overvejelser som fuldstændighed og præcision. Til dette formål præsenterer vi SHACTOR, et system skræddersyet til at udtrække og analysere formbegrænsninger og udnytte dem til at detektere og rette fejlagtige og forket data i vidensgraferne.

For at imødekomme udfordringen med dataadgang udnytter vi validerende former til at optimere behandlingen af SPARQL-forespørgsler i vidensgrafer. Vi foreslår "formstatistik"til estimering af kardinalitet under forespørgselsplanlægning, hvilket resulterer i effektiv forespørgselsoptimering.

Vi demonstrerer effektiviteten af denne tilgang på både syntetiske og virkelige datasæt og viser dens potentiale til at forbedre præstationen af forespørgslerne i store vidensgrafer.

Derudover introducerer vi S3PG, en ny tilgang med det formål at forbedre datainteroperabiliteten mellem to populære datamodeller for vidensgrafer: RDF og egenskabsgraf. S3PG sikrer tabsløs og monoton transformation, hvilket bidrager til at forbedre datainteroperabiliteten i vidensgrafer ved at muliggøre problemfri udveksling af data mellem forskellige vidensgrafdatamodeller.

Denne afhandling bidrager dermed til at fremme state-of-the-art inden for forskningen af vidensgrafer ved at løse disse udfordringer gennem nye metoder og teknikker. Den fremmer den praktiske anvendelse af vidensgrafer i forskellige domæner, herunder, men ikke begrænset til, informationsudvinding, dataintegration og vidensopdagelse. De metoder og værktøjer, der udvikles og offentliggøres som forskningsartikler i forbindelse med denne afhandling, bidrager til igangværende bestræbelser på at forbedre kvaliteten, tilgængeligheden og interoperabiliteten af vidensgrafer og baner dermed vejen for deres omfattende vedtagelse og dybtgående indvirkning på forskellige industrier, især inden for kunstig intelligens.

Acknowledgments

First and foremost, I am grateful to Allah the Almighty for giving me the strength and the opportunity (through His constant blessings) to complete the journey of PhD.

I would like to express my sincere gratitude to my PhD supervisors, Katja Hose and Matteo Lissandrini, for always being there for me. They have served as my mentors, they have not only supervised me, in fact they have trained me to the best of my ability through their guidance, experience, and expert knowledge. Simply put, this would not have been possible without them. Thank you so much!

I dedicate my thesis and my PhD degree to my parents, especially my late father, who passed away during third year of my PhD journey in November 2022. It was his ardent wish for me to obtain a PhD degree, and I regret that he is not here to witness my graduation. Every evening, when I left the office, we used to talk and discuss the day's events. He was a constant source of motivation and strength for me to overcome the ups and downs of the entire journey. My mother, who did not have the opportunity to attend school and does not fully comprehend the concept of a PhD, has always been praying for my success. After my father, she said, *"Go and finish what you started and what your father wanted you to do. Don't worry about me!"* I am the first in our family to earn a PhD degree, and this accomplishment is due to the hard work of my parents, who moved from the village to the city to give their children the best of the education they could afford.

I am thankful to my family, especially my beloved wife, Ateeqa Altaf, for her constant support throughout this experience. I am aware that it has not been easy, yet she has always been by my side, offering assistance, comfort, and inspiration when needed. For that, I have immense respect for her; This thesis would not have been possible without her help and support. My daughter Zara was born during the second year of PhD, and my life changed completely when she came into my life. Seeing her after a long day always made my day! Special thanks to my parents-in-law, my brother, and my sisters for always being there for me, providing immense support and help.

Special thanks to Prof. Angela Bonifati, my supervisor and coauthor for

one of my research papers written during the study abroad period at Lyon 1 University in France. Additionally, I am thankful to Daniel Hernandez and Riccardo Tommasini for their valuable input in shaping my ideas. Both played a vital role in my development as a researcher, and I am very grateful to them for their time and help.

Thanks to my special friend and colleague, Emil Riis Hansen, who was always there for me to discuss, no matter what the topic was about. We shared the office for 3 years and I would like to say that he is the most intelligent person I have ever met. I am grateful for his immense support and help. I also want to extend my thanks to my friends Imran Riaz and Muhammad Naeem; they were always there for me whenever I needed them. Furthermore, I am thankful to my colleagues from the Data, Knowledge, and Web Engineering (DKW) group at Cassiopeia, Aalborg University, for creating a wonderful work environment. They have cultivated a great work environment that is enjoyable and stimulating. I also express my gratitude to the administrative staff at Aalborg University for their patience and help in responding to my numerous, and often unnecessary, queries. Lastly, my gratitude extends to funding agencies, particularly the Danish Council for Independent Research (DFF) and the Poul Due Jensen Foundation.

In loving memory of my father, Ghulam Rabbani. Abbu, you will always be missed!

Kashif Rabbani
Aalborg University, Saturday 4th May, 2024

This research was partially funded by the Danish Council for Independent Research (DFF) under grant agreement no. DFF-8048-00051B, the EU's H2020 research and innovation programme under grant agreement No 838216, and the Poul Due Jensen Foundation.

Contents

Thesis Details

Thesis Title:	Scalable Extraction and Adoption of Shapes for Improving Data Quality and Query Processing in Knowledge Graphs
Ph.D. Student:	Kashif Rabbani
Supervisor:	Prof. Katja Hose, Aalborg University & TU Wien Austria
Co-Supervisor	Assoc. Prof. Matteo Lissandrini, Aalborg University & University of Verona, Italy

The main body of this thesis consists of the following five papers.

[A] **Kashif Rabbani**, Matteo Lissandrini, Katja Hose, "SHACL and ShEx in the Wild: A Community Survey on Validating Shapes Generation and Adoption" *In Companion Proceedings of the Web Conference, WWW 2022*, doi. 10.1145/3487553.3524253

[B] **Kashif Rabbani**, Matteo Lissandrini, Katja Hose, "Extraction of Validating Shapes from very large Knowledge Graphs" *In Proceedings of the Very Large Databases, VLDB 2023*, doi. 10.14778/3579075.3579078.

[C] **Kashif Rabbani**, Matteo Lissandrini, Katja Hose, "SHACTOR: Improving the Quality of Large-Scale Knowledge Graphs with Validating Shapes" *In Proceedings of the 2023 International Conference on Management of Data, SIGMOD-Companion 2023*, doi 10.1145/3555041.358972.

[D] **Kashif Rabbani**, Matteo Lissandrini, Katja Hose, "Optimizing SPARQL Queries using Shape Statistics" *In Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021*, doi 10.5441/002/EDBT.2021.59.

[E] **Kashif Rabbani**, Matteo Lissandrini, Angela Bonifati, Katja Hose, "Lossless Transformation of Knowledge Graphs to Property Graphs using Standardized Schemas" *(Under Review)*.

This thesis has been submitted for assessment in partial fulfillment of the PhD degree. The thesis is based on the scientific papers that are listed above.

Parts of the papers are used directly or indirectly in the extended summary of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Technical Faculty of IT and Design at Aalborg University. The permission for using the published and accepted articles in the thesis has been obtained from the corresponding publishers with the conditions that they are cited and DOI pointers and/or copyright/credits are placed prominently in the references.

Part I

Thesis Summary

Chapter 1

Introduction

This thesis details efforts to improve the *quality* and *access* of data in knowledge graphs and make their underlying data models *interoperable*. This thesis is written as a collection of scientific papers. First, Section 1 examines the problem at hand and the motivation of using shapes to improve the *quality*, *access*, and *interoperability* of data in knowledge graphs. Then, Section 2 describes the contributions of the thesis, followed by Section 3 describing the structure of the thesis summary.

1 Background and Motivation

Over the past two decades, the Semantic Web [13] has witnessed a profound evolution by providing access to a vast and interconnected Web of Data [43] often referred to as Linked Open Data (LOD) [39, 40]. The Semantic Web has developed a range of standards, including the Resource Description Framework (RDF) [24] as the standard data model, SPARQL [23] as the standard query language for RDF, and other standards to define the structure, restrictions, and semantics of RDF data, such as RDFS [25], OWL [101], and SHACL [51].

Within this rich landscape of the Semantic Web, the concept of a Knowledge Graph (KG) [44] has emerged as a powerful embodiment of interconnected data and semantic relationships. KGs are graph-structured representations designed to capture the semantics relationships between entities, used as versatile tools for symbolically representing and integrating knowledge in a structured manner [71]. KGs leverage the standards established by the Semantic Web and offer a practical realization of the vision for the Web of Data [43]. KGs play a pivotal role in various domains by allowing information to be stored and analyzed using the graph model [46, 63, 85]. They can be modeled as property graphs (PG) [28] or with the RDF data

model. KGs modeled as PGs are often used mainly in applications, such as social networks, recommendation systems, and fraud detection [16]. While KGs modeled as RDF triples are often used in applications such as question answering, semantic search, and reasoning [12].

There has been a rapid increase in the number of openly published KGs, spanning diverse topics including life sciences (e.g., Bio2RDF [29]), general knowledge (e.g., DBpedia [10] and Wikidata [96]), and government data (e.g., US Government LOD [42]). This exponential growth is evidenced by the LOD Cloud [54], consisting of 1,314 distinct KGs and 16,308 interlinking connections. KGs are also a key component of modern businesses, providing structured data and factual knowledge that not only improve products but also improve their intelligence [63]. Microsoft's Bing KG, for example, contains around 2 billion entities and 55 billion facts, while Google's KG has 1 billion entities and 70 billion facts, helping with searching and answering questions by including general knowledge about the world. Meta, the home of the world's largest social graph, has approximately 50 million entities and 500 million facts, including details about music, movies, celebrities, and places people are interested in [63].

As applications of KGs continue to evolve, three key issues have become increasingly prominent: the *quality of data within KGs*, the *ability to quickly access data through efficient data access mechanisms*, and the *need for interoperability between different types of KG data models*. The first challenge involves ensuring the accuracy, reliability, and consistency of the information stored in the KG. The second challenge focuses on the efficient access of relevant data, emphasizing the importance of efficient query processing for real-time applications. Lastly, the challenge of interoperability involves data exchange and integration between RDF and property graph data models of KGs, to facilitate the exchange and use of information between these models. This thesis refers to the aforementioned challenges as the (i) Data Quality Challenge (DQC), (ii) Data Access Challenge (DAC), and (iii) Data Interoperability Challenge (DIC).

1. **Data Quality Challenge (DQC).** The DQC in RDF graphs arises from the intricate and diverse nature of contributing data sources, presenting challenges such as: *(i) data heterogeneity:* RDF graphs aggregate data from various heterogeneous sources, each with its unique structure, format, and quality standards. This integration, while flexible, introduces the risk of inconsistencies and inaccuracies when merging data from disparate sources. *(ii) data evolution:* RDF graph data evolve over time with additions and updates, necessitating robust mechanisms for validation and verification to manage ongoing data quality. *(iii) schemaless nature:* RDF's inherent schemaless quality provides flexibility but results in variations in data representation, making it challenging to enforce uniform data quality standards.

Validating shapes such as SHACL (Shapes Constraint Language) [51] and ShEx (Shape Expressions) [72], play a crucial role in addressing such data quality issues in RDF graphs. These shape validation mechanisms provide a systematic approach for defining and verifying the structure and integrity of RDF data. SHACL became a W3C standard in 2017 and enables the creation of shape definitions that articulate the expected structure and constraints of RDF data. ShEx, on the other hand, offers a concise and expressive language for specifying shapes, providing a flexible means of evaluating RDF data against predefined criteria [51]. By employing validating shapes, practitioners can establish a set of rules and constraints that data must adhere to, thereby enhancing the quality and reliability of information encapsulated in RDF graphs. These validation approaches help identify and rectify data quality issues, ensure that data comply with expected standards, and contribute to a more reliable and accurate representation within RDF graphs.

Although RDF data validation has been made possible through the use of validating shapes to improve the quality of the data, the current state of the art has limitations when it comes to defining shapes for graphs [77]. For instance, none of the existing approaches is able to extract shapes from very large graphs, and, more importantly, they are not able to output shapes that are of good quality [79].

2. **Data Access Challenge (DAC).** This challenge refers to the limitations associated with efficient access to RDF data, which is typically stored in the form of a collection of triples (*subject*, *predicate*, and *object*) in triple stores. SPARQL is a W3C-standardized query language for querying data from RDF graphs. It uses triple patterns for graph matching and compares variables in these triple patterns with graph's data during query evaluation. The result of the evaluation process is then expressed as a set of mappings that link a set of variables to nodes and edges in the graph. Furthermore, it provides advanced operators such as FILTER, AND, OPTIONAL, and UNION to create more complex queries. Optimizing SPARQL queries is essential due to the complexity of processing them. SPARQL queries often involve multiple triple patterns and complex filtering conditions, making them vulnerable to performance issues. As KGs grow in size and complexity, SPARQL queries with numerous joins can make accessing data challenging in terms of time and space complexity.

   Despite the progress made in improving the *efficiency* of querying triplestores, it is still a major challenge as the size of RDF graphs and the need to query them in more intricate ways is constantly growing [3]. There are numerous techniques to optimize SPARQL query processing; however, a recent survey [3] on "RDF stores and SPARQL engines to query KGs" points out that there are still unresolved issues to address in terms of query *dynamics*, *types*, and *volume*.

3. **Data Interoperability Challenge (DIC).** The term interoperability was introduced in the area of information systems. It is defined as the ability of two or more systems or components to exchange information and use the information that has been exchanged [33]. In the context of data management, interoperability is concerned with the support of applications that exchange and share information across the boundaries of existing databases [20]. Providing interoperability between data models, systems, and applications is a very concrete and pragmatic problem, which stems from the need for reusing existing systems and programs for building new applications [20]. Data and information interoperability are relevant to promote data exchange and integration [66], to have a common understanding of the meanings of the data [41], to allow reuse and sharing of information and knowledge [87], and to allow exploring the best features of different approaches and systems [67].
   The DIC in the context of KGs refers to the limitations encountered when achieving interoperability between the RDF and the property graph data models of KGs. There exist various approaches in the literature to enable interoperability between RDF and PG data models of KGs [8]; however, the most recent survey [28] in this direction concluded that no transformation method supports data and schema transformations between both models, preserving the capabilities of both.

Given the details of data *quality*, *access*, and *interoperability* challenges above, this thesis investigates each of these challenges with the aim of building upon the state of the art to improve data quality, access, and interoperability; thus optimizing the utilization of KGs and Semantic Web standards. To accomplish this, the first step involves addressing the challenge of data quality in KGs. This is achieved by conducting a state of the art and community survey to analyze the generation and adoption of validating shapes (SHACL and ShEx). The results of the survey showed the need to develop semi-automatic methods that can help users generate shapes from large KGs. Following the results of the survey, an efficient and scalable algorithm was proposed to extract quality shapes from very large RDF graphs. To facilitate the use of the proposed algorithm, a tool was developed to assist users in extracting shapes and identifying and rectifying data quality issues such as spuriousness and erroneousness in large RDF graphs. Next, the challenge of data access is addressed by utilizing validating shapes, specifically SHACL, for SPARQL query optimization. Finally, the challenge of interoperability is tackled through the proposal of a standardized schema-based RDF to PG data model transformation approach.

As such, by building upon state of the art, the goal is to deliver robust methodologies alongside efficient algorithms and advanced tools that can improve the quality of KGs' data while allowing users to efficiently access

it and take full advantage of the PG data model by making RDF and PG interoperable.

In the remainder of this section, an overview is provided of the contributions of the thesis as well as the structure of the thesis. Details on the state of the art and the individual contributions are given in Sections 2-6.

2 Contributions of the Thesis

The primary contributions of this thesis include the development of novel methodologies and tools aimed at enhancing the data *quality*, *access*, and *interoperability* of KGs. For each of the aforementioned challenges, specific objectives have been established. To address DQC, the objective is to *improve the data quality in KGs*. Similarly, for DAC, the objective is to *enhance query processing in KGs*. Lastly, to tackle DIC, the objective is to *improve data interoperability in KGs*. As this is a publications based thesis, Figure 1.1 has been included to illustrate how each paper (A-E) contributes to achieving these three objectives and the connections between the papers.

At first, **Paper A** [77] studies the *generation* and *adoption* of validating shapes by carrying out a community survey to evaluate the requirements of users (from industry and academia) who generate validating shapes such as SHACL and ShEx. To do this, an extensive survey of existing tools for extracting validating shapes and their features was conducted, followed by a comparison of the survey results. Lastly, an examination of how existing

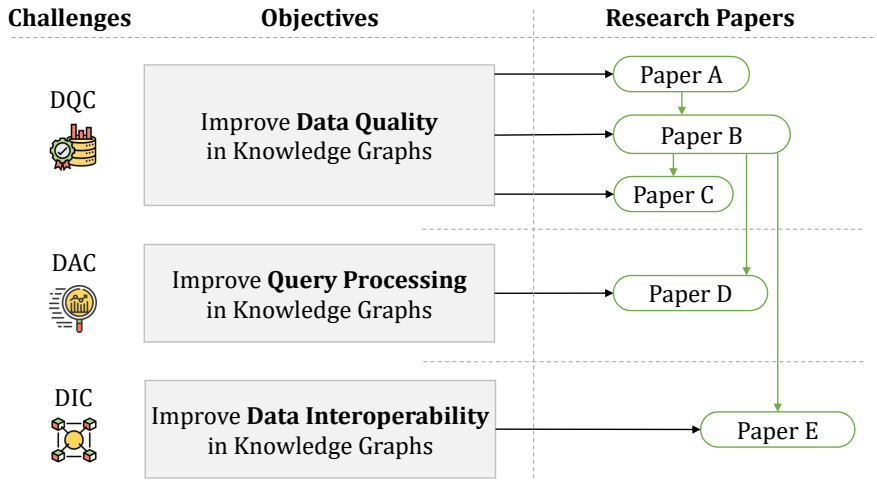


Fig. 1.1: Overview of challenges, objectives, and interconnections between research papers for enhancing data quality, access, and interoperability in KGs.

automatic shape extraction approaches work in practice on large real-world KGs was carried out. The analysis revealed the need for developing semi-automatic methods that can assist users in generating validating shapes for large KGs.

Paper B [79] delves deeper into the extraction of validating shapes and initially identifies the shortcomings of existing shapes extraction techniques, such as being *incapable of extracting shapes with all the constraints*, *not scalable*, and *prone to generating spurious shapes*. To address the limitations of existing approaches, this paper introduces Quality Shapes Extraction (QSE), an efficient and scalable approach for extracting quality shapes from very large KGs. Both exact and approximate variants of QSE are provided. This method offers information about the quality of shape constraints by calculating their confidence and support within a KG, enabling the identification of shapes that are most informative and less likely to be affected by incomplete or incorrect data. QSE was employed to extract SHACL shapes from large KGs such as DBpedia [10] and Wikidata [96]. QSE provides a considerable decrease in extraction time, i.e., almost 12 times faster than existing methods. Additionally, it filters out up to 93% of the invalid and spurious shapes, leading to a decrease of up to two orders of magnitude in the number of constraints presented to the user, e.g., in DBpedia, it reduced the number of constraints on properties from 11,916 to 809.

In **Paper C** [80], a system called SHACTOR is developed to facilitate the use of QSE [79]. SHACTOR is designed to improve the data quality in very large KGs by extracting and analyzing validating shapes. Utilizing standard shape extraction techniques often results in the generation of numerous shapes, some of which may stem from erroneous data in the KG. SHACTOR addresses this issue by using QSE to parse the given KG, comprising tens of millions of triples and thousands of classes. QSE offers an efficient and scalable shape extraction algorithm that outputs SHACL shape constraints annotated with statistical information, such as support and confidence. This facilitates the identification of erroneous and missing triples in the KG. SHACTOR uses such annotated shape constraints to help users identify and rectify errors by automatically generating SPARQL queries on the graph to pinpoint nodes and facts that cause incorrect shapes, allowing for data corrections.

Validating shapes not only play a pivotal role in mitigating quality issues in KGs but also extend their utility to various applications. **Paper D** [76] contributes to the objective of *improving query processing in KGs* to deal with DAC by making use of validating shapes. SPARQL query optimization involves *cardinality estimation* and *join ordering*. The traditional methods for SPARQL query optimization rely on global statistics over entire RDF graphs and thus have limitations to accurately capture common correlations of RDF KGs. This leads to erroneous estimations and suboptimal query execution plans, and it

is computationally expensive to capture correlations in a fine-granular manner. To mitigate this, Paper D introduced 'shapes statistics', an extension of the SHACL that incorporates statistical information to capture correlations between classes and properties. Then, these "shapes statistics" are used for cardinality estimation during query planning for join ordering in an open-source query engine. Lastly, the proposed query optimization method is evaluated on synthetic and real-world datasets, and the results revealed that the proposed approach is efficient in terms of both the pre-processing steps to generate shape statistics and the cardinality estimation to optimize query plans.

Paper E [75] significantly contributes to DIC by improving data interoperability in RDF KGs by leveraging validating shapes. This paper introduces a novel approach *Standardized SHACL Shapes-based PG Transformation (S3PG)* to enhance data interoperability, focusing specifically on the transformation of KGs from the RDF data model to the property graph (PG) data model using standardized schemas. S3PG employs SHACL for RDF data and PG-Schema [6] for PGs. PG-Schema [6] is established as the most recent standard published in 2023 and serves as a foundational element in this transformation process. S3PG is a completely lossless and monotonic transformation approach designed for the transformation of large RDF graphs. It preserves the information and semantics throughout the transformation process. S3PG was evaluated using DBpedia and a domain-specific KG representing clinical trials. The quality of the transformed graphs was assessed by executing various types of queries, the results indicated that S3PG always achieved a remarkable 100% accuracy rate, surpassing existing techniques having accuracy ranging from from 30% to 99%. Moreover, S3PG exhibits full monotonicity when dealing with evolving graphs and demonstrates a significantly reduced time requirement for incorporating changes compared to existing methods. The findings show the efficacy and superiority of S3PG in ensuring accurate, lossless, and efficient transformations in the context of RDF to PG conversions, and thus contribute to the objective of improving data interoperability in KGs.

3 Structure of the Thesis

The thesis has two parts. In **Part I**, Section 1 discusses the motivational aspects and fundamental contributions of the thesis. Section 2 discusses related work for data quality, query optimization, and interoperability in KGs. Next, the thesis provides a summary of each paper: Section 3 offers a comprehensive summary of the survey results presented in Paper A. Section 4 provides an overview of the proposed Quality Shapes Extraction (QSE) approach with core contributions from Paper B. Section 5 summarizes the essential aspects

of the SHACTOR system from Paper C to help users extract, analyze, and clean validating shapes. Section 6, presents a summary of the proposed approach for optimizing SPARQL queries through SHACL shape statistics, as detailed in Paper D. Lastly, Section 7 provides a summary of the S3PG approach, detailed in Paper E, which focuses on transforming RDF data models to PG data models using standardized schemas. Section 8 concludes the **Part I** by offering a forward-looking perspective on potential future directions stemming from the research work done in this thesis.

Part II of the thesis contains the full versions of all the papers, with the layout adjusted to fit the thesis format. It is suggested to read them in chronological order as shown in Figure 1.1.

Chapter 2

State of the Art

Knowledge Graphs (KGs) are modeled using Resource Description Framework (RDF) [24] and Property Graphs (PGs) [44]. As repositories of information continue to expand, KGs confront the escalating challenge of maintaining high-quality data and undergoing thorough validation [72, 77] to ensure their reliability in practical applications. In addition, KGs play a crucial role in real-time applications, introducing further challenges associated with efficient data access. The growing volume of data within KGs requires robust solutions for seamless and timely access. Moreover, KGs are not restricted only to the RDF data model; they are also represented using the PG data model. This diversity underlines the need for interoperability between RDF and PG data models, highlighting the importance of developing approaches that facilitate smooth interactions between these data models.

In this section, the state-of-the-art approaches to each of these aspects are analyzed, highlighting their advantages and shortcomings.

1 Data Quality in Knowledge Graphs

KGs data often originate from diverse sources, resulting in incomplete, duplicate, contradictory, or incorrect statements [44]. After initial creation and enrichment, assessing the quality of the KG data becomes a crucial step. Quality, in this context, pertains to fitness for purpose, determining the reliability of the KG data for specific uses. The assessment encompasses various dimensions and metrics, transitioning from traditional database quality considerations to those specific to KGs [44]. The discussion includes qualitative aspects captured by quality dimensions inspired by the framework presented by Batini and Scannapieco [57]. These dimensions [44] are discussed below.

- **Accuracy** in a KG pertains to how well entities and relations, repre-

sented by nodes and edges, faithfully represent real-world phenomena. This dimension includes syntactic accuracy, semantic accuracy, and timeliness.
- **Coverage** involves preventing the omission of domain-relevant elements to avoid incomplete query results, biased models, etc.
  For example, one aspect of coverage is *completeness*, which evaluates the presence of all the necessary information in a dataset. It comprises schema completeness, property completeness, population completeness, and linkability completeness. Measuring completeness is challenging. It often requires comparing with standards or evaluating recall from extraction methods based on complete sources.
  Another aspect of coverage is *representativeness*, which involves assessing high-level biases included or excluded in the KG.
- **Coherency** assesses how well a KG adheres to formal semantics and schema-level constraints. It involves *consistency*, which ensures the absence of logical contradictions, and *validity*, which ensures the absence of constraint violations, as captured by shape expressions.

Graph schemata are employed to define a high-level structure and/or semantics that the graph must adhere to or should adhere to in order to guarantee the correctness, coverage, consistency, and validity of data in graphs. Three types of graph schemata are defined in the literature [44], namely, Semantic Schema, Validating Schema, and Emergent Schema.

- **Semantic schema** allows for defining the meaning of high-level terms in a graph, facilitating reasoning. RDF Schema (RDFS [25]) is a prominent standard for this purpose, enabling the definition of sub-classes, sub-properties, domains, and ranges, serialized as a graph.
- **Validating schema** is defined using shapes to enable data completeness and validity. Shapes target nodes in a graph and specify constraints, supporting the restriction of values on properties. Shape Expressions (ShEx [72]) and Shapes Constraint Language (SHACL [51]) are two emerging shape languages for RDF graphs.
- **Emergent schema** is extracted as latent structures of graphs, often known as a graph summary [19]. Latent structures in a data graph can be automatically identified using frameworks like quotient graphs, which partition nodes based on an equivalence relation while preserving structural properties [70].

The scope of this thesis covers the dimensions of *coverage* and *coherency* for data quality assessment in KGs using *validating schema*. The following discusses the state-of-the-art related to validating schemas.

1.1 Data Validation

Integrity constraints for KGs were initially formulated using the RDF schema vocabulary [25] and subsequently with the OWL language [58, 59, 93]. The SPARQL Inferencing Notation (SPIN) [50] emerged as an alternative later on. SHACL [51], recognized as the next generation of SPIN, has been a W3C standard since 2017. Additionally, ShEx [73], although not standardized, serves as a constraint language using regular bag expressions inspired by XML schema languages and is employed in projects like WikiData [94]. While SHACL and ShEx are not entirely equivalent [32], they share a fundamental concept. Both enforce specific constraints on the combination of node types and predicates, ensuring each node satisfies these constraints [30].

Shapes Extraction:

The prevalence of large-scale KGs has led to the development of various applications to extract information related to their implicit or explicit schemas [49]. In particular, when it comes to constructing or extracting shapes, the goal is to generate a set of shapes from existing KG data. This process is essential for validating schemas that guarantee the quality of a KG's data. The results of the survey [77] demonstrate a growing need among practitioners for efficient methods to extract validating shapes from large existing KGs. Table 2.1 categorizes existing approaches based on several technical features, such as the ability to extract shapes from data or ontologies, the automation of shape extraction, compatibility with shapes extraction from a SPARQL triplestore, and the capability to extract SHACL, ShEx, or both types of validating shapes. It is noteworthy that there are approaches for extracting schemas from property graphs [52]. Nevertheless, their direct applicability to RDF KGs is restricted due to the intricate nature of RDF schema

Table 2.1: State-of-the-art to extract validating shapes (reproduced from [77])

Approach	*Extracted from*		*Automatic*	*Triplestore*	*Type*
	data	*ontology*			
Shape Induction [56]	✔	✗	✔	✔	SHACL, ShEx
SheXer [30]	✔	✗	✔	✔	SHACL, ShEx
Spahiu et al. [88]	✔	✗	✔	✔	SHACL
ShapeDesigner. [15]	✔	✗	✔	✔	SHACL, ShEx
SHACLGEN [48]	✔	✔	✔	✔	SHACL
TopBraid [95]	✔	✔	✔	✔	SHACL
Pandit et al. [65]	✗	✔	✗	✔	SHACL
Astrea [22]	✗	✔	✔	✗	SHACL
SHACLearner [64]	✔	✗	✔	✗	SHACL
Groz et al. [35]	✔	✗	✔	✗	ShEx

structures. Moreover, these techniques mainly concentrate on recognizing sub-types based on node labels (which are not present in RDF data, as types are nodes in the graph), and they lack a design specifically tailored to tackle the issue of spuriousness in RDF KGs.

Shapes Validation:

After defining validating shapes, the subsequent step involves their utilization for validating the RDF KGs. As of now, SHACL validation has been seamlessly integrated into various mainstream tools and triplestores [98]. A well-known example is RDF4J [81], a Java framework for RDF data management, which now incorporates a SHACL validation engine. RDF4J is a core component in several projects, notably the GraphDB [34] triplestore. Other databases equipped with SHACL capabilities include AllegroGraph [4] by Franz Inc, Apache Jena [47] by Apache, and Stardog [89] by Stardog Union Inc. A comprehensive benchmark for comparing different SHACL implementations, along with results for four distinct databases, was proposed in [84]. Python users can avail themselves of a SHACL implementation through the pySHACL [74] library. Pioneering the validation of recursive SHACL graphs, SHACL2SPARQL [26] stands as one of the initial tools. Conversely, Trav-SHACL [31] is a tool that implements a SHACL engine optimized for evaluating core constraints expressed in language fragments [27]. Demonstrating significantly faster validation times on these SHACL fragments, Trav-SHACL outperformed the SHACL2SPARQL tool. Notably, MagicShapes [2] represents a recent approach for unrestricted SHACL validation.

1.2 Data Refinement

In addition to quality evaluation, various techniques are available for the refinement of KGs, specifically for (semi)automatic completion and correcting them [69]. KG completion involves filling in missing edges and addressing the inherent incompleteness of KGs, while KG correction focuses on identifying and removing existing incorrect edges. Unlike the processes of creation and enrichment, refinement typically enhances KG without extensive reliance on external sources. These refinement methods, namely *completion* and *correction*, primarily target aspects of precision, coverage, and coherence. Although other quality issues, such as succinctness, could be considered, completion and correction tasks have dominated attention in KG refinement [69]. For a more in-depth exploration of the state of the art in KG refinement, it is recommended to read Paulheim's survey [69].

1.3 Summary

Given the current state of the art in assessing and refining data quality in KGs, this thesis contributes to the dimensions of *Coverage – completeness* and *Coherency – validity*. Specifically, a survey is conducted to analyze the adoption of validating schemas in the community. Based on the survey results, an approach called Quality Shapes Extraction (QSE) is proposed to extract validating schemas (in the form of SHACL) from very large KGs. In contrast to existing approaches, QSE is *efficient*, *scalable*, and produces *high-quality* shape constraints. QSE provides a notion of quality in shape constraints by computing the support and confidence of node and property shape constraints while mining shapes from large RDF graphs. The extracted validation schema can be used to validate very large KGs, thus contributing to the dimensions of *Coherency – validity*. Building upon the proposed approach, a system has been developed that not only facilitates the extraction of validating schema but also assists users in identifying and rectifying data-related issues. Thus, the proposed system contributes to the dimension of *Coverage – completeness* and refinement. Detailed information on these contributions is provided in Section 2.

2 Query Optimization in Knowledge Graphs

SPARQL [23] serves as the standard language for querying RDF KGs, supporting relational algebraic operations like joins, projection, selection, union, difference, etc. The introduction of features such as *property paths* in the latest version, SPARQL 1.1, enables the matching of paths of arbitrary length in RDF. The Web [13] hosts numerous popular SPARQL endpoints, processing millions of queries per day [53, 83]. Engines designed to store, index, and process SPARQL queries over RDF are commonly known as SPARQL engines. According to [3], these engines are also considered RDF stores due to their support for joins in SPARQL. RDF stores, or SPARQL engines, encompass critical components such as data storage, indexing, and join processing, which play an essential role in enhancing the performance of SPARQL engines [3].

- **Storage.** Various engines adopt diverse structures (e.g., tables, graphs), encodings (e.g., integer IDs, string compression), and storage media (e.g., main memory, disk) for RDF data. The choice of storage depends on factors such as data scale and supported query features [82].
- **Indexing.** RDF stores utilize indexes to perform search and query execution. Different index types cater to distinct operations, presenting varying time–space trade-offs.

- **Join Processing.** Efficient join processing is fundamental for query evaluation. It involves optimizing join ordering, which is crucial for ensuring computational efficiency. Cardinality estimation is one of the methods to optimize join ordering, in addition to that, recent innovations include multi-way joins, worst-case optimal joins, and GPU-based join processing.

To align with the scope of this thesis, the state of the art of *join processing* in SPARQL query engines is discussed, with a particular focus on *cardinality estimation*.

2.1 Join Processing and Cardinality Estimation

The study of cardinality estimation has been a focal point in the realm of relational databases [68]. In the context of SPARQL queries, existing techniques often adapt methodologies from the relational domain [45, 91], primarily focusing on specific query types [61]. Typically, these approaches construct various types of single or multidimensional synopses over databases to estimate cardinalities [90]. However, algorithms designed for generating synopses for unlabelled graphs are inapplicable to RDF graphs due to the labeled nature of edges. Consequently, existing approaches for RDF summaries either yield excessively large summaries [90], exhibit high computational complexities, or fail to preserve the RDF schema during summary construction [90]. Therefore, the most promising approaches involve leveraging statistics derived from edge-label frequencies.

In particular, RDF-3X [62] presents a method for cardinality estimation relying on edge label frequencies, employing a histogram-based approach. This method is extended by incorporating statistical information from Characteristic Sets [61], which compute frequencies of sets of predicates sharing the same subject to estimate cardinalities. While effective for star-shaped queries, it suffers from significant underestimation in the general case due to the independence assumption [68]. Attempts to address this limitation, such as Characteristic Pairs [55], are constrained to supporting multi-chain star queries. Additionally, extracting Characteristic Sets from large heterogeneous graphs proves computationally expensive. SumRDF [90], an approach based on graph summarization, faces challenges handling large queries due to prohibitive computation costs. Furthermore, constructing such summaries over extensive RDF graphs is resource-intensive [68]. A recent benchmark, G-CARE [68], thoroughly assessed the effectiveness of current methods for estimating cardinality in subgraph matching scenarios. Results revealed that approaches utilizing sampling and optimized for online aggregation demonstrate superior performance compared to conventional cardinality estimation methods specifically crafted for RDF graphs. This underlines the need for a

more thorough investigation into appropriate cardinality estimation methods for SPARQL query optimization.

In recent work, Shape Expressions (ShEx) [72] have been employed to reorder triple patterns, facilitating SPARQL query optimization [1]. This optimization estimates the order of execution for triple patterns based on heuristic inferences regarding their selectivity. For example, if a shape definition implies that every instructor has one or more courses, while every course has exactly one instructor, the optimization infers that the cardinality of courses is at least the same as that of instructors and likely larger. Notably, this optimization procedure is not grounded in actual data.

2.2 Summary

In the context of improving join processing in SPARQL queries over RDF KGs, this thesis introduces a novel contribution to improve join ordering. Our approach focuses on proposing a new method for estimating cardinalities, distinct from existing methodologies. Unlike conventional approaches that rely on global statistics for cardinality estimation [37], this method leverages fine-grained statistics derived from SHACL shapes. This allows us to achieve more precise cardinality estimations for query planning. In contrast to the practice of creating extensive and costly summaries and characteristic sets on RDF graphs to estimate cardinalities, this approach involves the use of SHACL shape constraints, which are as expressive as ShEx [72]. Compared to alternative solutions, this method entails lightweight preprocessing and preserves the structure of the original RDF graph and the SHACL validation schema. Detailed information on these contributions is provided in Section 2.

3 Interoperability in Knowledge Graphs

In recent years, a great deal of effort has been devoted to exploring data exchange and data integration. Nevertheless, there have been few attempts to address the issue of interoperability between RDF and Property Graph (PG) data models. In the following, the state of the art for transforming RDF data models into PG data models and vice versa is discussed.

3.1 RDF Data Model to Property Graph Data Model

In the domain of graph data representation, the transformation of RDF data models into PG data models has gained attention. Angles et al. [9] contributed direct mappings for this conversion, encompassing both data and schema aspects. Their work introduces schema-dependent and schema-independent direct mappings for transformation, utilizing RDFS as the

schema for KGs and a custom-defined schema for PGs. In particular, the authors contend that the property graph data model subsumes the information capacity of the RDF data model.

To address the limitations of SPARQL in the implementation of traversal or analytics algorithms, G2GML [21] provides a mapping of RDF graphs to PG. It introduces an exchangeable serialization format to enhance support for different graph database management systems and ensure interoperability. However, it also involves a redefined PG model. Despite these advances, the process of transforming RDF data models into PG data models is not without challenges. In particular, such transformations often result in incomplete data [28]. Haihong et al. [38] proposed an approach specifically tailored for the transformation of RDF to PGs in Hugegraph. The methodology addresses challenges such as ensuring the uniqueness of nodes and supporting multi-label and empty-label RDF graphs in single-label graph databases. However, it is crucial to note that this approach is specific to Hugegraph and may not be universally applicable. Furthermore, the authors caution that information preservation is not guaranteed during the import process, as the system automatically identifies vertices and edges labels, omitting blank nodes along with their related edges.

In a separate effort, rdf2neo [17] is introduced as a tool to populate Neo4j databases from RDF datasets. This tool exemplifies its utility through real use cases in agrigenomics, demonstrating how it can enhance opportunities for knowledge sharing and interoperability.

3.2 Property Graph Data Model to RDF Data Model

Exploring the reverse transformation from PGs to KGs involves several notable works. PREC [18] facilitates the user-configured conversion of property graphs to RDF graphs with a focus on more effective capture of semantic content. In the broader landscape, some work proposes unified storage layouts for RDF and PG data models [7, 102]. In [7], the authors advocate for a unifying data model that supports graph formats such as RDF, RDF*, and PGs. They introduce a flexible data model for graphs, with the aim of accommodating these diverse features. However, it is essential to acknowledge potential limitations in leveraging the individual capabilities of RDF and PG models within such unified models.

3.3 Summary

In the context of achieving interoperability between RDF and PG data models, this thesis introduces a novel approach to transform RDF data models into PG data models by utilizing standardized schemas for both data models. In contrast to existing methods, the proposed Standardized SHACL

Shapes-based PG transformation technique (S3PG) incorporates the standard SHACL schema for RDF graph data models and the PG-Schema (published in 2023) [6] as the schema for PG data models. This schema fulfills fundamental requirements, including the definition of node and edge types, as well as addressing advanced scenarios like expressing complex type hierarchies and integrity constraints. To the best of our knowledge, S3PG is the only lossless approach for transforming RDF to PG data models, ensuring the preservation of information, semantics, and monotonicity throughout the transformation process.

Chapter 3

Generation and Adoption of Validating Shapes

This section gives an overview of Paper A [77].

1 Motivation and Problem Statement

As discussed in Section 1, the popularity of Knowledge Graphs (KGs) has witnessed a significant surge over the past decade. However, these KGs, mainly represented in RDF, often lack data due to the ever-changing nature of human knowledge. Ensuring the accuracy of information within KGs is crucial. Validation languages such as SHACL [51] and ShEx [72] play a vital role in defining constraint rules (in the form of shapes, also referred to as validating shapes) that data must follow. These rules help maintain data quality and have applications in various areas like user interface design and query optimization [43, 76]. Several tools [15, 22, 30, 48, 95] are available to assist in creating these validation rules. However, manually defining rules for large KGs can be daunting, while automatic methods may produce an overwhelming number of rules that require careful validation.

To better understand user needs in shaping large KGs, in Paper A, we conducted a comprehensive online survey involving both academic and industry professionals. The survey aimed to uncover common approaches used in rule creation, identify challenges, and suggest future research directions in automating rule generation and validation processes.

2 Survey Methodology and Results

We conducted a comprehensive survey using Google Forms and shared it within the Semantic Web and KG communities, targeting members from various platforms and conferences. The survey, conducted anonymously between November 2021 and January 2022, yielded thirty responses, with a breakdown showing 53% from Industry, 27% from Academia, and 20% from both sectors. The survey delved into the methods employed for generating validating shapes within KGs. Results indicated that manual shape generation is the most prevalent method, followed by the utilization of existing ontologies and derivation from RDF graph instance data (as shown in Figure 3.1). Moreover, respondents reported a diverse array of tools and methodologies, with TopBraid Composer [95] being utilized by 33% of respondents, Protégé [30] by 20%, RDFShape [100] by 16.7%, SheXer by 10%, and text editors by 16.7%.

Further analysis revealed insights into the characteristics of the graphs for which respondents generated shapes. A significant proportion of respondents worked with graphs containing varying numbers of triples, classes, and distinct properties. Interestingly, while a majority of respondents generated shapes for entire graphs, a substantial portion also focused on specific portions, driven by various factors outlined in their responses.

In summary, the survey highlighted the prevalent use of manual shape generation despite the diverse range of tools and methodologies available. The results underscored the need for scalable tools to facilitate efficient shape generation, especially for large and complex graphs.

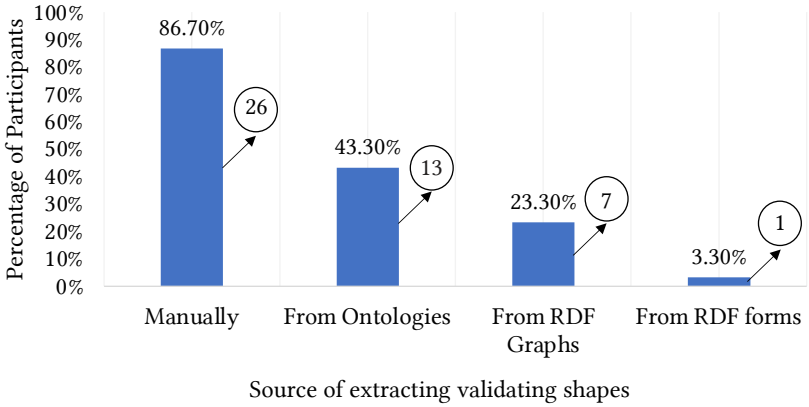


Fig. 3.1: Analysis on extraction of validating shapes (reproduced from Paper A [77]).

3 Limitations and Opportunities

The section elaborates on the disparities observed between the current capabilities of tools for shape extraction and the genuine requirements of users. Despite the availability of automated tools, manual shape generation remains prevalent among users, indicating a gap between tool capabilities and user needs. This necessitates further research avenues for improving automated shape extraction methods.

Existing approaches for shape extraction, as discussed in Section 1.1, vary in their support for extracting validating shapes. Some tools focus solely on ontology-based shape extraction, while others concentrate on instance data. However, few tools offer support for both ontology and instance-based shape extraction. These approaches often make assumptions that diverge from real-world scenarios, such as presuming the completeness of ontologies or the scalability of instance data processing [22, 30, 48].

To assess the actual capabilities of existing tools, we conducted experiments on state-of-the-art tools like SheXer [30], ShapeDesigner [15], and SHACLGEN [48]. These experiments were performed on datasets including DBpedia [10], YAGO-4 [92], and a scaled version of LUBM [36]. The results of these experiments revealed limitations in the scalability and reliability of these tools. For instance, ShapeDesigner encountered difficulties with datasets containing a few million triples, while SHACLGEN struggled with datasets featuring hundreds of classes, requiring significant time for shape extraction. Furthermore, the utility and reliability of shapes extracted from instance data were examined. It was found that shapes extracted by existing tools often lacked comprehensiveness and reliability, particularly for non-literal predicate objects. Manual inspection of these shapes revealed inadequacies in capturing all necessary constraints, indicating reliability issues. To address these concerns, efforts were made to extract missing shape constraints and assess their support within the graph. However, due to the inherent noise and incompleteness of KGs, automatically generated shapes and constraints often lack reliability.

4 Conclusion

In Paper A, the limitations found in existing shape extraction approaches align with the prevalence of manual shape generation among users, as indicated by the community survey (Figure 3.1). This shows the need for further research to assist users in generating useful validating shapes for existing large-scale KGs. Therefore, in the paper titled 'Extraction of Validating Shapes from very large Knowledge Graphs' (Chapter 4), we introduce a scalable and quality shapes extraction approach to overcome this challenge and provide automated support for shape generation in large-scale KGs.

Chapter 4

Quality Shapes Extraction

This section gives an overview of Paper B [79].

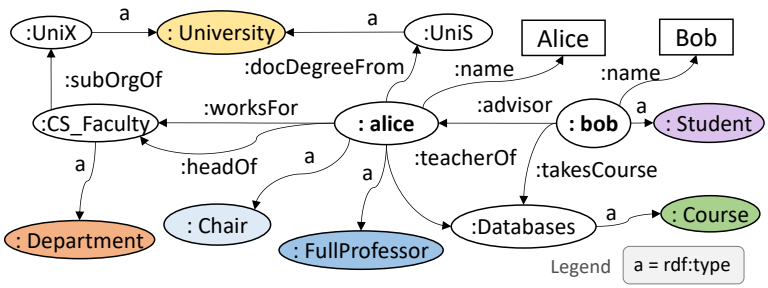
1 Motivation and Problem Statement

Building upon the results of our community survey, we investigate the limitations in existing approaches for extracting validating shapes and present a solution to overcome these limitations. Specifically, when defining validating shapes for a given KG, we can express that an entity of type Student requires a name, a registration number, and should be enrolled in some courses; and that these attributes should be of type string, integer, and Course, respectively – see Figures 4.1a and 4.1b for an example KG and corresponding shapes. The existing automatic [22, 30, 48, 56] and semi-automatic [15, 65, 95] techniques[1] to extract validating shapes suffer from three important limitations:

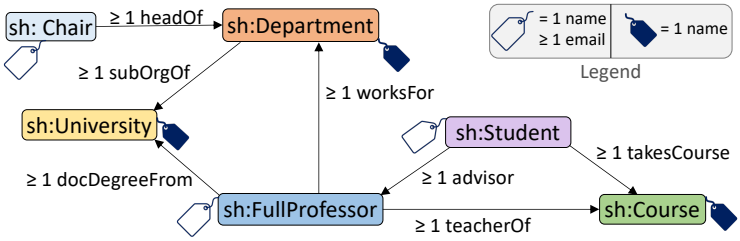
1. they are not able to produce complete shapes, e.g., they can identify that a student should have a property of type takesCourse but they do not extract the fact that the object should be of type Course;
2. the shapes they produce are easily affected by errors and inconsistencies in the KG, e.g., if some departments, by mistake, are attached the property hasAdvisor, a corresponding *spurious shape* is extracted; and
3. they do not scale to large KGs, e.g., they cannot process the full English WikiData, and require days to process its subset.

Therefore, in Paper B, we present the first technique for *efficient extraction of validating shapes from very large existing KGs that also ensures robustness against the effects of spuriousness*. Spuriousness poses important challenges to automatic shape extraction methods. For instance, in DBpedia [11], some entities

[1]Note: All these techniques are discussed in Chapter 2: State of the Art.



(a) An example RDF graph showing the academic connections between student Bob and full professor Alice. Alice serves as Bob's advisor, teaches a Databases course (taken by Bob), and is associated with the Computer Science Faculty at University X. Additionally, Alice holds a doctoral degree from University Y and holds leadership positions as head and chair of the department.



(b) A visual representation illustrating shape constraints for the RDF graph described above, applying constraints at both the class level (using Node Shapes) and the property level (using Property Shapes), including cardinality constraints.

Fig. 4.1: An example RDF Graph and its Validating Shapes — (adapted from [79])

representing musical bands are wrongly assigned to the class dbo:City. As a consequence, when shapes are extracted from its instance data using existing approaches, the resulting node shape for dbo:City specifies that cities are allowed optional properties like dbo:genre and dbo:formerBandMember. Hence, due to the effect of *spuriousness*, existing approaches generate tens of thousands of shapes (our experiments show that standard extraction processes produce more than 2 million property shapes for WikiData [97]). Thus, domain experts cannot manually identify valid shapes. Therefore, to tackle the issue of *spuriousness*, we study and formalize the problem of *support-based shapes extraction* and propose the Quality Shapes Extraction (QSE) approach as a solution to this problem. To tackle the issue of *scalability*, we devise two efficient algorithms, QSE-Exact and QSE-Approximate. *Hence,* QSE *can filter out shapes affected by spurious or erroneous data based on robust and easily understandable measures.*

2 Methodology

The standard model for encoding KGs is RDF (Resource Description Framework) [24], where data is represented as triples denoting subject, predicate, and object. An RDF graph consists of nodes and edges, with nodes representing IRIs, blank nodes, or literals. Predicate IRIs are categorized into predicates and classes, with the type predicate linking entities to their classes.

A validating shape schema represents integrity constraints over a KG, describing constraints on node types and their properties. This schema is defined as node shapes and property shapes in SHACL.

To analyze the reliability of extracted shapes, we introduce support and confidence metrics for shape constraints. Support counts the number of entities satisfying a shape constraint, while confidence measures the proportion of entities conforming to a constraint among all instances of the target class.

2.1 QSE-Exact

QSE-Exact aims to extract shapes from RDF graphs stored in files or triplestores. It involves four phases: entity extraction, entity constraints extraction, support/confidence computation, and shapes extraction. The algorithm iteratively processes triples to extract entity and property information, compute support and confidence, and generate shape constraints. In the file-based approach, QSE-Exact processes RDF graphs stored in files, extracting entities, constraints, and shape information iteratively from the input file. It involves parsing triples, extracting entity types, and computing support and confidence for shape constraints. The query-based variant of QSE-Exact retrieves entity and property information from triplestores using SPARQL queries. It computes support and confidence for shape constraints using count queries, enabling shape extraction from triplestores.

QSE-Exact assigns cardinality constraints to property shapes based on entity confidence levels. Mandatory properties are inferred based on high confidence, while incomplete KGs allow users to specify confidence thresholds for adding constraints. The time complexity of QSE-Exact is linear with respect to the size of the RDF graph and the number of shapes extracted, making it not very suitable for large-scale KGs due to high memory consumption.

2.2 QSE-Approximate

QSE-Approximate is introduced as a solution to address the memory consumption issue faced by QSE-Exact, particularly when dealing with large KGs. The main novelty of QSE-Approximate lies in its utilization of dynamic reservoir sampling, which allows for efficient extraction of validating shape schemas while reducing memory requirements. QSE-Approximate operates

by maintaining separate reservoirs for each distinct entity type in the KG. These reservoirs dynamically adjust their sizes as new triples are processed, ensuring that a representative sample of entities is retained while minimizing memory consumption. The algorithm iterates over the RDF graph, parsing triples and filtering them based on type declarations and property assertions.

During the entity extraction phase, entities are identified, and their corresponding entity types are recorded. Each entity type maintains a reservoir, and entities are added to the appropriate reservoir based on their type. When a reservoir reaches its capacity limit, QSE-Approximate employs a dynamic node replacement strategy to ensure that the reservoir contains a diverse set of entities. This strategy involves selecting a random entity from the reservoir and replacing it with the current entity, taking into account the scope of the node (i.e., the number of types associated with it). This approach ensures that the reservoirs maintain a balanced representation of entity types while efficiently utilizing available memory.

Once the sampling phase is complete, QSE-Approximate utilizes the sampled data to estimate the actual support and confidence of shape constraints for pruning based on the user's provided thresholds for minimum support and confidence.

3 Evaluation and Discussion

We evaluate the effectiveness of our proposed solutions in addressing spuriousness within KGs and compare them with existing methods.

We selected a mix of synthetic and real-world datasets, including LUBM-500 [36], DBpedia [11], YAGO-4 [92], and WikiData [97] (Wdt15 [99] and Wdt21). All experiments were conducted using QSE algorithms implemented in JAVA-11 on a machine with 16 cores and 256 GB RAM. The source code is available on GitHub[2], and the extracted SHACL shapes are published on Zenodo [78]. For comparison, we used SheXer [30]. We evaluated QSE based on metrics such as running time, memory usage, and shape statistics. Through these evaluations, we aimed to provide insights into the performance and efficacy of our proposed solutions in addressing spuriousness in KGs.

Evaluation of QSE-Exact. We use QSE-Exact for shape extraction from LUBM, DBpedia, YAGO-4, and WikiData. Statistics on the shapes extracted using QSE-Exact (file-based) are presented in Table B.2 (in Paper B), covering Node Shapes (NS), Property Shapes (PS), and Property Shape constraints (PSc). Initially, state-of-the-art approaches like SheXer, ShapeDesigner, and SHACLGEN were considered for comparison with our approach. Due to limitations in handling large KGs, only SheXer was suitable for comparison

[2] https://github.com/dkw-aau/qse

against QSE. Table B.3 (in Paper B) shows the running time and memory consumption for shape extraction using file-based (F) and query-based (Q) variants of SheXer, QSE-Exact, and QSE-Approximate. QSE-Exact outperforms SheXer, being faster and more memory-efficient across all datasets. While SheXer faces memory limitations and timeouts for some datasets, QSE-Exact manages these challenges more effectively, timing out only for WikiData. To mitigate spuriousness, we prune extracted shapes based on support and confidence thresholds. Increasing these thresholds leads to higher percentages of pruned PSc and PS. For example, with a confidence threshold >25% and support ≥ 1, QSE prunes 99% of PSc and PS in DBpedia and significant portions in Wdt21.

Evaluation of QSE-Approximate. QSE-Approximate reduces memory demands compared to QSE-Exact by allowing users to set sampling percentage (S%) and maximum reservoir size (τ_{max}). It outperforms QSE-Exact and SheXer in efficiency (see Table B.3 in Paper B). For instance, extracting shapes from Wdt21, QSE-Approximate (with $\tau_{max} = 1000$ and S%=100%) was nearly twice as fast and used one-tenth the memory of QSE-Exact, while SheXer failed. Overall, QSE-Approximate addresses scalability issues in shape extraction. QSE-Approximate's output quality is evaluated on Wdt21 with varied sampling percentage (S%) and max reservoir size τ_{max} values while keeping confidence and support thresholds constant. Results (shown in Table B.4 of Paper B) indicate that S%=10 and τ_{max} up to 200 achieve 92% precision for PS, requiring 16 GB RAM and 81 minutes. S%=50% and $\tau_{max} = 5K$ on a 24 GB RAM machine yield 96% precision in 95 minutes. On a 32 GB RAM machine, S%=100% and $\tau_{max} = 5K$ result in 100% precision in 98 minutes. Additionally, the impact of pruning on shapes extracted from Wdt21 using QSE-Approximate is analyzed with different confidence and support values (Table B.5). With support ≥ 1 and confidence >25%, QSE-Approximate achieves nearly all PS extracted by QSE-Exact for Wdt21, with 89% recall and 100% precision, demonstrating minimal underestimation of support and confidence.

Practical Implications of QSE. We assess QSE's practicality by verifying the accuracy of shapes extracted from DBpedia and their role in KG validation. Using QSE with confidence >25% and support >100, we extracted shapes and manually inspected 10 for correctness. The analysis showed 100% precision in shape constraints, removing spurious ones. Validating these shapes with a SHACL validator on DBpedia revealed 20,916 missing triples and 155 erroneous triples. This experiment highlights our technique's ability to offer users accurate shapes for efficiently identifying KG errors.

4 Conclusion

In Paper B, we presented an automated method for extracting shapes, known as Quality Shapes Extraction (QSE), to tackle the common problems of scalability and spuriousness found in current methods. QSE offers both exact and approximate solutions for efficient shape extraction on commodity machines. By leveraging support and confidence metrics, data scientists can prioritize shapes with high reliability, aiding in resolving data quality concerns. Even with conservative pruning thresholds, QSE significantly reduces the number of shapes, up to 93%, compared to trivial extraction methods. These pruned shapes, lacking substantial support, are likely to be spurious. Additionally, our study demonstrates that the approximate technique incurs minimal loss in the quality and completeness of extracted shapes.

To better support data scientists in creating and validating shapes, as well as assisting users responsible for maintaining data quality in KGs, paper titled 'SHACTOR: Improving the Quality of Large-Scale Knowledge Graphs with Validating Shapes' (Chapter 5) presents a system based on QSE. This system enables users to automatically extract validating shapes, identify errors, highlight missing data, and generate queries for error correction.

Chapter 5

Improving Data Quality using Shapes

This section gives an overview of Paper C [80].

1 Motivation and Problem Statement

Quality Shapes Extraction (QSE) [79] extracts validating shapes from large graphs on commodity machines, providing information about the reliability of the extracted shape constraints using confidence and support metrics. QSE identifies informative shapes while distinguishing those affected by incomplete or incorrect data. Building upon the same motivation, in Paper C, we demonstrate how shapes extracted with QSE, along with confidence and support information, enable various data profiling and cleaning functionalities beyond simple validation. We introduce *SHACTOR (SHapes extrACTOR)*, a tool that accelerates the KG cleaning process by automatically extracting shapes, evaluating their quality, providing structural profiling information, identifying errors, highlighting missing data, and generating SPARQL queries to correct issues. SHACTOR leverages QSE to filter shapes by confidence and support thresholds, highlighting spurious shapes and facilitating interactive data cleaning and shape correction.

2 SHACTOR

SHACTOR provides a user-friendly interface for interacting with our QSE algorithm [79], offering various additional functionalities. It utilizes QSE (Paper B) to extract validating shapes from an RDF graph, enabling explo-

ration and analysis. SHACTOR comprises three phases: (1) shape extraction with support and confidence, (2) shape analysis, and (3) KG cleaning. Support measures entities conforming to a specific constraint, while confidence quantifies the ratio of entities conforming to the total instances of the target class. SHACTOR aims to enhance data quality in RDF graphs by assisting users in generating high-quality validating shapes for analysis and correction, mitigating spuriousness effects in shape extraction.

Shapes Extraction and Analysis. SHACTOR parses RDF graphs to extract entities and constraints, computes support and confidence for each constraint, and allows the selection of a custom subset of types for analysis. It employs the QSE algorithm to produce shapes meeting specified thresholds, enabling dynamic filtering and fine-tuning. Interactive visualization of shape distribution and quality indicators aids in identifying noisy or incomplete data. Moreover, SHACTOR generates SPARQL queries to retrieve entities and triples violating given constraints, facilitating data inspection and identification of erroneous or missing information. Users can execute queries to enhance KG quality by deleting erroneous triples or inserting missing data.

SHACTOR exemplifies its effectiveness in leveraging shapes annotated with statistical data to expedite the KG cleaning process. We illustrate how data scientists can utilize SHACTOR to automate shape extraction and assess the quality of extracted shapes alongside their provenance. The demonstration begins by guiding participants through various extraction phases, utilizing a full snapshot of DBpedia from 2021, comprising 52 million triples, 15 million literals, 5 million typed entities, 1.3 thousand properties, and 427 classes. Additionally, participants can analyze shapes from a full snapshot of WikiData, containing over 1.9 billion triples.

Shapes Analysis. Users input support and confidence thresholds for analysis. SHACTOR applies these thresholds and presents an overview using pie charts (① and ② in Figure 5.1), aiding users in selecting optimal threshold values. Quality indicators for node shapes are displayed, and users can explore individual shape constraints (e.g., :CityShape) and associated property shapes (④ in Figure 5.1). Property constraints below the thresholds are highlighted, facilitating the identification of spurious shapes.

Finding Errors in the KG. SHACTOR assists in identifying spurious shape constraints by highlighting constraints with low support or confidence. Users can generate SPARQL queries to retrieve triples associated with these shapes (⑤) and view retrieved triples (⑥). Users can then take corrective actions, such as deleting erroneous triples or modifying classifications.

Finding Missing Information in the KG. SHACTOR allows users to explore property shapes, identifying object IRIs lacking types and entities missing properties. Based on this analysis, SHACTOR suggests INSERT queries to

add missing information, such as adding values for specific properties to entities missing them.

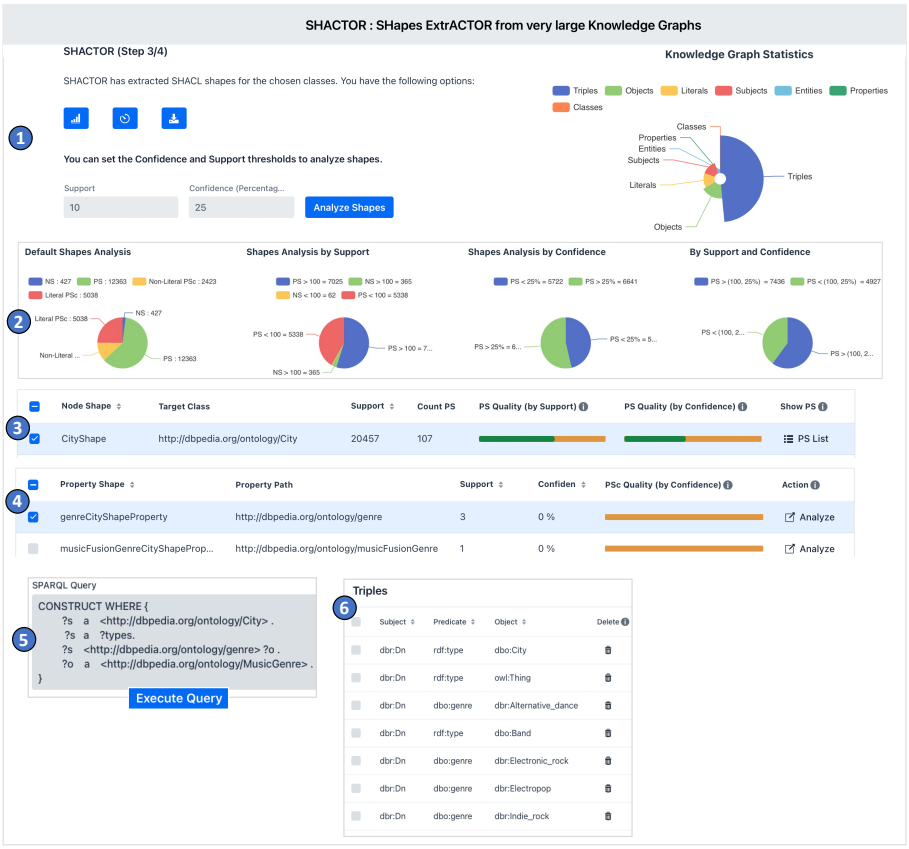


Fig. 5.1: Sample Analysis of Shapes for DBpedia using SHACTOR GUI – (adapted from [80])

3 Conclusion

In paper C, we introduced SHACTOR, a system designed to facilitate end-to-end profiling and cleaning of large-scale KGs by leveraging validating shapes automatically extracted from the graph and enriched with statistical information via our scalable QSE presented in paper 'Extraction of Validating Shapes from very large Knowledge Graphs' (Chapter 4). Moreover, this demo highlights the versatility and effectiveness of employing shapes as easily extractable and user-friendly tools for pinpointing and rectifying data quality issues in existing KGs.

Next, we discuss how shapes are used to enhance query optimization in KGs as of paper 'Optimizing SPARQL Queries using Shape Statistics' (Chapter 6) and achieve interoperability in KGs as of paper 'Lossless Transformation of Knowledge Graphs to Property Graphs using Standardized Schemas' (Chapter 7).

Chapter 6

Optimizing Query Processing using Shapes

This section gives an overview of Paper D [76].

1 Motivation and Problem Statement

Validating shapes not only plays a pivotal role in mitigating quality issues in KGs but also extends their utility to various applications. For instance, when querying RDF KGs using SPARQL query language, existing query optimization approaches encounter challenges in finding efficient query plans. We investigate the use of validating shapes to optimize query processing in SPARQL query processing in Paper D. Existing approaches (discussed in Section 2) to SPARQL query optimization rely on global statistics derived from the entire RDF graph. However, these methods often fail to capture the inherent correlations within RDF KGs accurately. Consequently, such limitations can result in inaccurate estimations and suboptimal query execution plans. Moreover, the computational overhead of capturing correlations at a fine-grained level further intensifies these challenges.

To mitigate these issues, in Paper D, we introduce "shapes statistics" as an extension of SHACL validating shapes. This approach allows to capture correlations between classes and properties within KGs using shape constraints. Leveraging *shapes statistics*, we improve cardinality estimation during query planning, explicitly focusing on join ordering within an open-source query engine. Our evaluation, conducted on synthetic and real-world datasets, underscores the efficiency of our proposed methodology. Notably, our approach streamlines the pre-processing steps necessary for generating shape statistics and facilitates optimized cardinality estimation, leading to better query plans.

2 Methodology

RDF graphs represent entities and relationships as triples <*subjects, predicates, objects*>. An RDF graph consists of triples $\langle s, p, o \rangle$ where s and o can be IRIs, blank nodes, or literals. SPARQL is a query language for RDF data. A SPARQL query consists of triple patterns and conditions. Each triple pattern can have concrete or variable elements. SHACL is a schema language for RDF. A SHACL shapes graph defines constraints on RDF data using node and property shapes.

In the context of SPARQL queries, the problem entails devising an optimal plan to execute a given query Q by determining the most efficient order in which to join its triple patterns. This optimization seeks to minimize the overall execution cost of the query. To achieve this, it is necessary to estimate the join cardinalities between pairs of triple patterns in Q, and subsequently arrange them in the join order that minimizes the total execution cost. Thus, the problem consists of two main tasks: first, estimating join cardinalities for all pairs of triple patterns in Q, and second, optimizing the join order based on these estimates to minimize the total execution cost.

Extending Shapes with Statistics. To improve join cardinality estimations, we extend SHACL's node and property shapes with detailed statistical insights from the RDF graph. These statistics, termed *shapes statistics* include total triple counts, minimum and maximum triple counts per instance, and the number of distinct objects for property instances. Computed through SPARQL queries, these statistics enhance the understanding of RDF data. Additionally, we introduce *global statistics* by enriching VOID statistics with precise RDF property insights, such as distinct subject and object counts. While adaptable to OWL and RDF Schema, our focus remains on SHACL shapes for their structured simplicity.

Estimating Triple Pattern Cardinality. Each SPARQL query involves joins between multiple triple patterns, necessitating the estimation of matching triples for each pattern. We utilize statistical insights from the extended SHACL shapes graph to derive these estimates. By associating triple patterns with their corresponding node or property shapes, statistical information like *sh:count* and *sh:distinctCount* guides the estimation process. Formulas derived from previous research [37] summarized in Table D.1 of Paper D aid in computing the expected cardinality based on *global* statistics and *shapes* statistics.

Estimating Join Cardinality. Joins occur between two triple patterns sharing a common variable, categorized as Subject-Subject (SS), Subject-Object (SO), or Object-Object (OO) join. We estimate these join cardinalities using equations D.1, D.2, and D.3 in Paper D, incorporating distinct subject and object counts, i.e., DSC_i and DOC_i from the triple patterns.

Determining Join Order. Given an RDF graph and its statistics graphs, we propose an algorithm to compute the optimal join ordering for a query Q. Initially sorting triple patterns by their estimated cardinalities using global statistics, we then refine this ordering by considering shapes statistics. The algorithm iteratively selects triple patterns with the least estimated join cardinality, ultimately yielding an optimized join order. Tables D.2a and D.2b in Paper D present join orderings computed using global and shapes statistics, respectively, demonstrating the impact of statistical insights on query optimization.

3 Evaluation and Discussion

We evaluated the performance of query plans generated by our algorithm using both global and shape statistics, comparing them to plans from Apache Jena ARQ and GraphDB query engines, as well as RDF cardinality estimation methods including Characteristic Sets [61] and SumRDF [90]. Experiments were conducted on a single machine running Ubuntu 18.04 with 16 cores and 256GB RAM.

Datasets. We assessed query plan performance across different datasets and sizes: LUBM [36], WatDiv [5], and YAGO-4 [92]. Specifically, we used LUBM-500, two WatDiv variants (WATDIV-S with ~108.9 million triples and WATDIV-L with 1 billion triples), and a YAGO-4 subset linked to English Wikipedia articles.

Implementation. We introduced a *Shapes Annotator* to extend SHACL shapes graphs with statistics. In cases where shapes are absent, existing shape extraction methods (explained in Section 2) can be employed. We implemented our join ordering algorithm in Java using Jena[3], with the source code available on our website[1].

Data Loading and Query Planning Time. We loaded all three datasets and their corresponding SHACL shapes graphs into Jena TDB, employing our join ordering algorithm to construct query plans utilizing global and shape statistics. For Jena, we used its ARQ query engine to derive the plans. Datasets were also loaded into GraphDB, utilizing its *onto:explain* feature to obtain query plans. For the Characteristic Sets [61] approach, characteristic sets were generated for each dataset, with Extended Characteristic Sets [55] used to optimize non-star type queries. Generating characteristic sets for large RDF graphs incurs significant computational overhead. For instance, it took 6.2 hours to generate sets for LUBM, 1.2 hours for WATDIV-S, and 8.2 hours for YAGO-4. Similarly, for SumRDF [90], summaries were generated for each

[1] https://relweb.cs.aau.dk/rdfshapes/

dataset, with their generation time varying based on size and heterogeneity. For example, it took 4.5 minutes to summarize LUBM, 14 minutes for WATDIV-S, and 4.3 hours for YAGO-4. To ensure reasonable times, parameters for the target summary size matched those in SumRDF [90]. For YAGO-4, a target size of 100K was selected. Query planning time for all approaches consistently remained below 20 milliseconds, with subsequent analysis focusing on cardinality estimation precision and query performance.

Query Classification. Queries were classified into complex, snowflake, and star categories. LUBM offered 14 default queries with simple structures, from which we selected five queries and supplemented additional queries for complex, snowflake, and star patterns. The WatDiv benchmark consisted of three complex, sever star, and five snowflake queries. Due to the lack of standard queries or query logs for YAGO-4, we manually formulated 13 queries based on complex, snowflake, and star queries graph patterns from the WatDiv Benchmark, accessible on our website.

Query Runtime. For LUBM, query runtime analysis depicted that plans proposed by the Shape Statistics approach outperformed Global Statistics for queries with at least one type-defined triple pattern. GS plans were competitive with those of GraphDB, Characteristic Sets, and SumRDF, with Characteristic Sets demonstrating inadequacy for large snowflake queries. Conversely, plans proposed by Jena often proved suboptimal and non-deterministic due to its heuristics-based query optimizer. Similarly, query runtime for YAGO-4 showed competitive performance of Shape Statistics and Global Statistics plans against those of GraphDB, Characteristic Sets, and SumRDF, with variations observed in snowflake queries.

4 Conclusion

In Paper D, we introduced a novel approach to improve cardinality estimation for optimizing SPARQL queries in RDF KGs. Our methodology includes introducing shape statistics to capture RDF graph correlations, a novel cardinality estimation technique, and an efficient join ordering algorithm. Through extensive experimentation on synthetic and real-world datasets, we have demonstrated the efficiency of our approach compared to two prominent SPARQL query engines and two state-of-the-art RDF cardinality estimation methods. The outcomes highlight the effectiveness of our method in preprocessing for statistical analysis and estimating cardinality to optimize query plans.

Chapter 7

Improving Interoperability in Knowledge Graphs using Shapes

This section gives an overview of Paper E [75].

1 Motivation and Problem Statement

Validating shapes not only plays a pivotal role in mitigating quality issues in KGs but also extends their utility to various applications. For instance, existing approaches (as discussed in Section 3) result in data loss when transforming KGs modeled as RDF data models to Property Graph (PG) data models. Therefore, we investigate the use of validating shapes to enable lossless transformation from RDF to PG data models in Paper D.

Specifically, in Paper D, we introduce a novel approach to enhance data interoperability, focusing specifically on the transformation of KGs from the RDF data model to the PG data model using standardized schemas. The proposed technique, Standardized SHACL Shapes-based PG Transformation (S3PG), employs SHACL for RDF data and PG-Schema [6] for PGs. PG-Schema [6] is established as the most recent standard published in 2023 and serves as a foundational element in this transformation process. S3PG is a completely lossless and monotonic transformation approach designed for the transformation of large RDF graphs. It maintains the information and semantics during the transformation process and consistently achieves a 100% accuracy rate when compared to existing transformation methods.

2 Methodology

As RDF and validating shapes have been explained earlier, we omit to define them again. A property graph is a directed attributed multi-graph where nodes and edges are labeled and have attributes [6]. It consists of sets of labels, property names, property values, and records, where a record maps keys to values. A property graph is a tuple containing sets of nodes and edges, functions mapping edges to node pairs, nodes and edges to labels, and nodes and edges to records [6].

PG-Schema [6] provides a standardized schema definition for property graphs comprising PG-Types and PG-Keys. PG-Types define node and edge types based on allowed label and content combinations. At the same time, PG-Keys enforce constraints on typed data, including integrity constraints such as keys, participation, and cardinality. PG-Schema supports inheritance between types, allows intersections and unions for content types, and can define abstract node types.

Schema and Data Transformation Problems. The problem of transforming an RDF KG into the corresponding property graph is divided into two subproblems: schema transformation and data transformation. Given the shape schema of an RDF graph, the *schema transformation problem* aims to generate a corresponding PG-Schema. This involves creating a schema transformation function that maps node and property shape constraints of shape schema to PG-Types and PG-Keys in PG-Schema, ensuring that PG-Schema accurately represents all constraints of PG-Schema. *Data transformation* involves converting RDF graphs into PG while conforming to their schemas and the transformation function. The objective is to produce a function that maps graph nodes and edges to PG elements, ensuring the resulting PG conforms to the PG schema.

Transformation properties. The literature identifies a set of desirable properties for data transformation [86], including information preservation, query preservation, semantic preservation, and monotonicity. *Information preservation* ensures that the original information in the RDF graph can be recovered from the transformed PG, facilitated by computable mappings. A transformation is deemed *query preserving* if any query over the source graph yields the same result when evaluated over the transformed graph PG. This property relies on the existence of equivalent SPARQL queries. *Semantic preservation* guarantees that equivalent constraints enforced on the source graph can also be applied to the transformed target graph, ensuring that all constraints of the shape schema are reflected in the resulting PG-Schema. Finally, *monotonicity* ensures that when new data is added or deleted, adjustments are made only to the corresponding data in the target graph without requiring re-computation of the entire transformation [86].

2.1 S3PG: Schema Transformation

While transforming SHACL shape schema to PG-Schema, S3PG schema transformation function iterates through sh:core constraints and transform all node and property shape constraints of a given shape schema to elements in PG-Schema. Node shapes specify types (sh:targetClass or sh:node) and properties (sh:property). For instance, in Figure 7.1, shape:Person defines a :name property with data type string and cardinality [1,1], and shape:Student inherits properties from shape:Person. These shapes are converted to node types in PG-Schema as shown in Figure 7.2. Property shapes, defined by sh:path, indicate paths, cardinality, and node kinds. Cardinality constraints [min,max] are translated into PG-Schema based on the connected node type. For example, a cardinality of [1,1] for a literal target results in a mandatory property. Cardinalities influence the transformation process, e.g., [1,1] implies a mandatory property. Non-literal properties are represented as edge types between nodes in PG-Schema. Properties linking to multiple literal values are transformed into node types for each data type. Properties with multiple non-literal values become edge types capable of accepting various node types. Properties accommodating both literal and non-literal targets are handled by creating node types for non-literal types.

```
shape:Person
 rdf:type       sh:NodeShape;
 sh:property
     [ sh:path :name;
       sh:nodeKind sh:Literal ;
       sh:datatype xsd:string;
       sh:minCount 1;
       sh:maxCount 1   ] .
 sh:targetClass :Person.
```

(a) Person node shape with :name property shape

```
shape:Student
 rdf:type        sh:NodeShape;
 sh:property
     [ sh:path :regNo;
       sh:nodeKind sh:Literal ;
       sh:datatype xsd:string;
       sh:minCount 1;
       sh:maxCount 1   ] .
 sh:targetClass  :Student;
 sh:node         shape:Person.
```

(b) Student Node Shape with :regNo property shape

Fig. 7.1: SHACL syntax for Person and Student – (adapted from Paper E)

```
(personType:  Person {name STRING})
```

(a) Person Node type with name property

```
(studentType: Student {regNo STRING}),
(studentType: studentType & personType).
```

(b) Student Node Type with regNo property and inheriting name property from Person Node Type

Fig. 7.2: PG-Schema syntax for Person and Student SHACL shapes defined in Figure 7.1 – (adapted from Paper E)

The schema transformation method encodes single-value properties as key-value pairs within nodes when possible to be memory efficient. However, this approach may not remain entirely schema monotone when schemas evolve. To address this, our approach adopts a non-conservative model, ensuring schema monotonicity by representing properties like :regNo of node

shape shape:Student as edge types, even for simple key-value pairs. This adaptation enables the PG-Schema to accommodate evolving schemas while maintaining schema monotonicity.

2.2 S3PG: Data Transformation

After schema transformation, the next step is to address the graph data transformation problem. S3PG offers two graph data transformation models: the *parsimonious* and the *non-parsimonious*. The key difference lies in how single-value properties are modeled in the PG data model. In the parsimonious model, properties are encoded as key-value attributes within nodes whenever possible, leveraging PG modeling capabilities. Conversely, the non-parsimonious model represents all property data as nodes, ensuring adaptability to any structural changes in the source graph and preserving the monotonic property of the transformation.

The transformation of an RDF graph into a property graph using PG-Schema involves processing the triples in RDF graph and analyzing the types of nodes involved. We propose a two-phase algorithm to solve this problem, where entities are first extracted and transformed into nodes in PG, followed by the parsing of property information to create edges and attributes. The algorithm begins by extracting entities and their types from RDF graph and then iterates over these entities to create PG nodes. In the second phase, it processes triples from RDF graph, creates edges between nodes based on the triples, and handles the transformation of property data into edges or key-value pairs within nodes. The resulting PG can be stored in any file format for loading into a graph database. S3PG ensures data transformation monotonicity by adapting its approach to handle literal properties consistently with other properties. This ensures full monotonicity despite any structural changes in the schema during data transformation.

The time complexity of S3PG's data transformation algorithm depends on the input graph size, the number of entities extracted, and the set of allowed labels size in the target graph. Initialization and entity transformation take linear time, while property transformation has a time complexity proportional to the product of the input graph size and the label set size. Overall, the time complexity is linear in the input size and the label set size.

3 Evaluation and Discussion

We compared the effectiveness and efficiency of S3PG with existing methods, using two versions of DBpedia [11] and the Bio2RDF [14] Clinical Trials dataset. The datasets were chosen to explore scalability and domain-specific aspects [11, 14]. Our experiments, conducted on a machine with 16 cores and

256 GB RAM running Ubuntu 18.04, benchmarked S3PG against NeoSemantics [60] and rdf2pg [9].

S3PG, implemented in JAVA-17, demonstrated superior performance in schema and data conversion time, maximum memory usage, and graph loading time. We also assessed the quality of transformed graphs by executing queries and verifying semantic preservation through integrity constraint queries. Comparative analysis with NeoSemantics and rdf2pg revealed that S3PG achieved better overall transformation and loading times while remaining within memory limits. Notably, the graphs transformed by S3PG contained more nodes and edges compared to the other methods, attributed to its modeling approach for multi-type properties [11].

Quality Evaluation. We assessed the transformed PG data model's quality by querying the DBpedia2022 dataset using SPARQL queries converted into Cypher for each method. S3PG utilized PG-Schema, NeoSemantics its setup, and rdf2pg its schema-dependent model. Queries were categorized into four groups based on node shape constraints. S3PG achieved 100% accuracy for Single Type and multi-type homogeneous literal queries, outperforming NeoSemantics and rdf2pg. All methods scored 100% for multi-type homogeneous non-literal queries. For multi-type heterogeneous queries, S3PG consistently achieved 100% accuracy, surpassing NeoSemantics (90.48% to 99.99%) and rdf2pg (30.22% to 99.99%).

Effect on Query Runtime. We compared query runtime between RDF and PG models transformed by S3PG, NeoSemantics, and rdf2pg on the DBpedia2022 dataset. Experiments were conducted using GraphDB and Neo4j as the respective DBMS. Results showed S3PG's efficient performance across various query types, especially in multi-type homogeneous queries. For multi-type heterogeneous queries, S3PG outperforms NeoSemantics and rdf2pg due to its optimized data representation. While SPARQL queries on RDF models are effective, S3PG's cypher queries exhibit superior performance in multi-type heterogeneous queries. Ultimately, the choice between SPARQL and Cypher depends on specific use cases and technical requirements.

Monotonicity Analysis. We demonstrated the monotonic behavior of S3PG, which offers parsimonious and non-parsimonious transformation models. Using snapshots of DBpedia from March 2022 (Dbp22march) and December 2022 (Dbp22dec), we compute Δ to represent changes between the two versions. The addition of $16.7M$ new triples and deletion of $5.9M$ triples are observed. Additionally, $16.1M$ triples were updated. Transforming Dbp22march using the parsimonious model took 34.0 minutes, and using the non-parsimonious model took 31.83 minutes. Transforming Dbp22dec from scratch with the parsimonious model took 34.25 minutes, approximately 7.59% longer than using the non-parsimonious model. Incorporating Δ into

Dbp22march using the non-parsimonious model took 9.97 minutes, resulting in a time savings of 24.28 minutes compared to the parsimonious model, representing a substantial 70.87% reduction in overall transformation time.

4 Conclusion

In Paper E, we introduced S3PG, a method for transforming RDF KGs into property graphs, addressing interoperability challenges between the two data models. S3PG utilizes SHACL for RDF and PG-Schema for property graphs, ensuring schema compliance, lossless transformation, and monotonicity. Evaluation using DBpedia 2022 and Bio2RDF clinical trial datasets demonstrates S3PG's superiority in efficiency compared to existing methods. Quality analysis confirms its reliability, achieving 100% accuracy across various query types on DBpedia. Moreover, S3PG maintains monotonicity when transforming evolving graphs, requiring minimal time to incorporate updates.

Chapter 8

Conclusions and Future Work

In conclusion, this thesis proposes novel methodologies to solve data quality, data access, and data interoperability challenges in Knowledge Graphs (KGs). The thesis consists of five research papers contributing towards the objective of solving each of the above-mentioned challenges. Given below, we summarize the contribution of each research paper.

- In Paper A [77], we study the generation and adoption of constraints on KGs in the form of validating shapes by carrying out a community survey. This paper contributes to the challenge of data quality as a step towards improving data quality in KGs. We conducted an extensive survey of the existing tools and methodologies for extracting validating shapes (such as SHACL or ShEx) and their features and then compared the results. We examined how existing automatic shape extraction approaches work on large real-world KGs. The analysis revealed the need for developing semi-automatic methods that can assist users in generating validating shapes for large KGs and provide a notion for the quality of data in the KGs.
- In Paper B [79], inspired by the results of the survey, we delve deeper into the extraction of validating shapes in the state-of-the-art and identify the shortcomings of existing shapes extraction techniques. We identified that existing approaches are *incapable of extracting complete shapes*, *not scalable*, and *prone to generating spurious shapes*. Therefore, to address these limitations, we introduced Quality Shapes Extraction (QSE), an *efficient scalable* shapes extraction approach to extract *quality* shapes from very large KGs. This paper contributes to the challenge of data quality to improve the data quality in KGs. It evaluates the quality of shape constraints by assessing their confidence and support within a KG, enabling the identification of highly informative shapes less susceptible to incomplete or erroneous data. We used QSE to extract SHACL shapes

from large KGs such as DBpedia and Wikidata. Results showed that QSE significantly reduces extraction time, nearly 12 times faster than existing methods. Moreover, it filters out approximately 93% of invalid and spurious shapes, resulting in a reduction of up to two orders of magnitude in the number of constraints presented to users. For example, in DBpedia, it reduced the number of property constraints from 11,916 to 809.

- In Paper C [80], we developed a system called SHACTOR to facilitate the use of QSE [79] and improve the data quality in very large KGs by extracting and analyzing validating shapes. Utilizing standard shape extraction techniques often results in the generation of numerous shapes, some of which may stem from erroneous data in the KG. SHACTOR addresses this issue by using QSE to parse the given KG, comprising tens of millions of triples and thousands of classes. QSE offers an efficient and scalable shape extraction algorithm that outputs SHACL shape constraints annotated with statistical information, such as support and confidence. This facilitates the identification of erroneous and missing triples in the KG. SHACTOR uses such annotated shape constraints to help users identify and rectify errors by automatically generating SPARQL queries on the graph to pinpoint nodes and facts that cause incorrect shapes, allowing for data corrections.
- In Paper D [76], we leveraged validating shapes to optimize SPARQL query processing in KGs. This paper contributes to the objective of *improving query processing in knowledge graphs* to deal with the data access challenge. Query optimization involves *cardinality estimation* and *join ordering*. We propose shapes statistics, as an extension of the validating shapes (SHACL) that incorporates statistical information to capture correlations between classes and properties of RDF graphs. Then, we use shapes statistics to estimate cardinality during query planning for join ordering in an open-source query engine. We evaluated our query optimization method on synthetic and real-world datasets, and the results revealed that our approach is efficient in terms of both the preprocessing steps to generate shape statistics and the cardinality estimation to optimize query plans in both synthetic and real-world large KGs.
- In Paper E [75], we leveraged validating shapes to improve interoperability in KGs. This paper presents a novel approach, S3PG, aimed at enhancing data interoperability in KGs by transforming RDF data models into Property Graph (PG) data models using standardized schemas. S3PG utilizes SHACL for RDF data and PG-Schema for PGs, ensuring lossless and monotonic transformation. Evaluation on DBpedia and Clinical Trials KG revealed S3PG's accuracy of 100%, outperforming existing techniques with accuracies ranging from 30% to 99%. Additionally, S3PG demonstrates full monotonicity and reduced time re-

quirements for incorporating changes, highlighting its effectiveness in improving data interoperability in KGs.

Future Work

The popularity of KGs has surged in recent years, leading to an increasing interest in initiatives such as approaches, methods, and technologies that ensure high data quality within these graphs. As the size of KGs expands rapidly, optimizing query processing to enable efficient data access is critical. And, KGs are commonly modeled using either the RDF or the property graph data model, requiring efforts to make them interoperable and facilitate seamless exchange and integration of data.

In this thesis, we have presented methodologies as a step towards improving data quality, optimizing query processing, and enhancing data interoperability in KGs. However, our work also highlights some interesting and important future research directions within these areas.

As noted in [79], there are certain limitations in our proposed approach to shape extraction. Therefore, as part of our future work, we are extending the QSE algorithm to provide theoretical guarantees on pruning when utilizing its approximate variant. Additionally, we are optimizing the QSE query-based algorithm to enhance performance in shape extraction when the graph source is a SPARQL endpoint. Moreover, there are several interesting directions for further enhancing the QSE algorithm, e.g., an automated solution is required to learn optimal configurations for pruning thresholds and sampling parameters for the approximate shape extraction process. Additionally, QSE can be adopted to design a technique that can learn constraints from data to output quality shape constraints. Further, after defining shapes, the next step is to make use of shapes to validate KG data. In this direction, approaches need to be designed that efficiently validate KG data and provide validation reports that are both human and machine readable. This will facilitate seamless integration into data management workflows and ensure accurate data quality control at scale.

Optimizing SPARQL query processing has been a topic of interest for decades. However, as highlighted in [76], there is a need for developing query optimizers that can consider optimizing SPARQL query operators such as OPTIONAL, FILTER, EXISTS/NOT EXISTS, UNION, and path queries. Our proposed shape statistics-based query optimization approach can be extended to optimize these query operators.

Interoperability in KGs' RDF and property graph data models has various applications, such as data integration and exchange. Our proposed transformation approach (S3PG in Paper E) currently only supports the transformation of KGs modeled as RDF data models to PG data models. Reverse trans-

formation requires separate treatment. Therefore, in the future, we plan to propose an algorithm that can transform property graph data models to RDF using PG-Schema and SHACL shapes. Additionally, an open question to tackle for future work is how to extend the SHACL specification to the RDF* data model and how to include it in the transformation algorithm of S3PG.

References

[1] A. Abbas, P. Genevès, C. Roisin, and N. Layaïda, "Selectivity estimation for sparql triple patterns with shape expressions," in *ICWE*, 2018, pp. 195–209.

[2] S. Ahmetaj, B. Löhnert, M. Ortiz, and M. Šimkus, "Magic shapes for shacl validation," *Proceedings of the VLDB Endowment*, vol. 15, no. 10, pp. 2284–2296, 2022.

[3] W. Ali, M. Saleem, B. Yao, A. Hogan, and A.-C. N. Ngomo, "A survey of rdf stores & sparql engines for querying knowledge graphs," *The VLDB Journal*, pp. 1–26, 2022.

[4] "AllegroGraph," https://allegrograph.com/, accessed: November 27, 2023.

[5] G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee, "Diversified stress testing of rdf data management." in *ISWC*, 2014, pp. 197–212.

[6] R. Angles, A. Bonifati, S. Dumbrava, G. Fletcher, A. Green, J. Hidders, B. Li, L. Libkin, V. Marsault, W. Martens, F. Murlak, S. Plantikow, O. Savkovic, M. Schmidt, J. Sequeda, S. Staworko, D. Tomaszuk, H. Voigt, D. Vrgoc, M. Wu, and D. Zivkovic, "Pg-schema: Schemas for property graphs," *Proc. ACM Manag. Data*, vol. 1, no. 2, pp. 198:1–198:25, 2023. [Online]. Available: https://doi.org/10.1145/3589778

[7] R. Angles, A. Hogan, O. Lassila, C. Rojas, D. Schwabe, P. Szekely, and D. Vrgoč, "Multilayer graphs: A unified data model for graph databases," in *Proceedings of the 5th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, 2022, pp. 1–6.

[8] R. Angles, H. Thakkar, and D. Tomaszuk, "Rdf and property graphs interoperability: Status and issues." *AMW*, vol. 2369, pp. 1–11, 2019.

[9] ——, "Mapping RDF databases to property graph databases," *IEEE Access*, vol. 8, pp. 86 091–86 110, 2020. [Online]. Available: https://doi.org/10.1109%2Faccess.2020.2993117

[10] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives, "Dbpedia: A nucleus for a web of open data," in *ISWC 2007*, 2007, pp. 722–735.

[11] ——, "Dbpedia: A nucleus for a web of open data," in *The Semantic Web, 6th International Semantic Web Conference*, ser. Lecture Notes in Computer Science, vol. 4825. Busan, Korea: Springer, 2007, pp. 722–735.

[12] C. Belth, X. Zheng, J. Vreeken, and D. Koutra, "What is normal, what is strange, and what is missing in an knowledge graph," in *The Web Conference*, 2020.

[13] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, vol. 284, no. 5, pp. 34–43, 2001.

[14] Bio2rdf, "Clinical trials," https://download.bio2rdf.org/#/current/clinicaltrials/, 2020, accessed on October 21, 2023.

[15] I. Boneva, J. Dusart, D. Fernández-Álvarez, and J. E. L. Gayo, "Shape designer for shex and SHACL constraints," in *Proceedings of the ISWC 2019 Satellite Tracks*, ser. CEUR Workshop Proceedings, vol. 2456. Auckland, New Zealand: CEUR-WS.org, 2019, pp. 269–272.

[16] A. Bonifati, S. Dumbrava, E. Martinez, F. Ghasemi, M. Jaffré, P. Luton, and T. Pickles, "Discopg: property graph schema discovery and exploration," *Proceedings of the VLDB Endowment*, vol. 15, no. 12, pp. 3654–3657, 2022.

[17] M. Brandizi, A. Singh, and K. Hassani-Pak, "Getting the best of linked data and property graphs: rdf2neo and the knetminer use case." in *SWAT4LS*, 2018.

[18] J. Bruyat, P.-A. Champin, L. Médini, and F. Laforest, "Prec: semantic translation of property graphs," in *1st workshop on Squaring the Circles on Graphs, SEMANTiCS*, 2021.

[19] Š. Čebirić, F. Goasdoué, H. Kondylakis, D. Kotzinos, I. Manolescu, G. Troullinou, and M. Zneika, "Summarizing semantic graphs: a survey," *The VLDB journal*, vol. 28, pp. 295–327, 2019.

[20] S. Ceri, L. Tanca, and R. Zicari, "Supporting interoperability between new database languages," in *[1991] Proceedings, Advanced Computer Technology, Reliable Systems and Applications*. IEEE, 1991, pp. 273–281.

[21] H. Chiba, R. Yamanaka, and S. Matsumoto, "G2gml: Graph to graph mapping language for bridging RDF and property graphs," in *Lecture Notes in Computer Science*. Springer International Publishing, 2020, pp. 160–175. [Online]. Available: https://doi.org/10.1007%2F978-3-030-62466-8_11

[22] A. Cimmino, A. Fernández-Izquierdo, and R. García-Castro, "Astrea: Automatic generation of SHACL shapes from ontologies," in *ESWC*, ser. Lecture Notes in Computer Science, vol. 12123. Heraklion, Crete, Greece: Springer, 2020, p. 497.

[23] W. Consortium, "Sparql 1.1 overview," https://www.w3.org/TR/sparql11-query/, 2013, accessed 23rd November, 2023.

[24] ——, "Rdf 1.1 concepts and abstract syntax," 2014.

[25] ——, "RDF Schema 1.1," https://www.w3.org/TR/rdf-schema/, 2014, accessed 17th January, 2023.

[26] J. Corman, F. Florenzano, J. L. Reutter, and O. Savkovic, "Shacl2sparql: Validating a sparql endpoint against recursive shacl constraints." in *ISWC (Satellites)*, 2019, pp. 165–168.

[27] J. Corman, J. L. Reutter, and O. Savković, "Semantics and validation of recursive shacl," in *The Semantic Web–ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part I 17*. Springer, 2018, pp. 318–336.

[28] D. Di Pierro, S. Ferilli, and D. Redavid, "Lpg-based knowledge graphs: A survey, a proposal and current trends," *Information*, vol. 14, no. 3, p. 154, 2023.

[29] M. Dumontier, A. Callahan, J. Cruz-Toledo, P. Ansell, V. Emonet, F. Belleau, and A. Droit, "Bio2rdf release 3: A larger, more connected network of linked data for the life sciences," in *ISWC 2014 Posters & Demonstrations Track*, vol. 1272, 2014, pp. 401–404.

[30] D. Fernandez-Álvarez, J. E. Labra-Gayo, and D. Gayo-Avello, "Automatic extraction of shapes using shexer," *Knowledge-Based Systems*, vol. 238, p. 107975, 2022.

[31] M. Figuera, P. D. Rohde, and M. Vidal, "Trav-shacl: Efficiently validating networks of SHACL constraints," in *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*. Ljubljana, Slovenia: ACM / IW3C2, 2021, pp. 3337–3348. [Online]. Available: https://doi.org/10.1145/3442381.3449877

[32] J. E. L. Gayo, E. Prud'Hommeaux, I. Boneva, and D. Kontokostas, "Validating rdf data," *Synthesis Lectures on Semantic Web: Theory and Technology*, vol. 7, no. 1, pp. 1–328, 2017.

[33] A. Geraci, *IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries*. IEEE Press, 1991.

[34] "Ontotext GraphDB," https://graphdb.ontotext.com/, accessed: November 27, 2023.

[35] B. Groz, A. Lemay, S. Staworko, and P. Wieczorek, "Inference of shape graphs for graph databases," in *25th International Conference on Database Theory, ICDT 2022, March 29 to April 1, 2022, Edinburgh, UK (Virtual Conference)*, ser. LIPIcs, vol. 220. Edinburgh, UK: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 14:1–14:20. [Online]. Available: https://doi.org/10.4230/LIPIcs.ICDT.2022.14

[36] Y. Guo, Z. Pan, and J. Heflin, "Lubm: A benchmark for owl knowledge base systems," *Journal of Web Semantics*, vol. 3, pp. 158–182, 2005.

[37] S. Hagedorn, K. Hose, K. Sattler, and J. Umbrich, "Resource planning for SPARQL query execution on data sharing platforms," in *International Workshop (COLD) co-located with the 13th ISWC)*, vol. 1264, 2014.

[38] E. Haihong, P. Han, and M. Song, "Transforming RDF to property graph in hugegraph," in *Proceedings of the 6th International Conference on Engineering MIS*. ACM, sep 2020. [Online]. Available: https://doi.org/10.1145%2F3410352.3410833

[39] A. Harth, K. Hose, and R. Schenkel, Eds., *Linked Data Management*. Chapman and Hall/CRC, 2014.

[40] O. Hartig, K. Hose, and J. F. Sequeda, "Linked data management," in *Encyclopedia of Big Data Technologies*. Springer, 2019.

[41] S. Heiler, "Semantic interoperability," *ACM Computing Surveys (CSUR)*, vol. 27, no. 2, pp. 271–273, 1995.

[42] J. A. Hendler, J. Holm, C. Musialek, and G. Thomas, "US government linked open data: Semantic.data.gov," *IEEE Intell. Syst.*, vol. 27, no. 3, pp. 25–31, 2012.

[43] A. Hogan, *The Web of Data*. Springer, 2020.

[44] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier *et al.*, "Knowledge graphs," *ACM Computing Surveys (Csur)*, vol. 54, no. 4, pp. 1–37, 2021, extended version.

[45] H. Huang and C. Liu, "Estimating selectivity for joined RDF triple patterns," in *CIKM*. Glasgow, United Kingdom: ACM, 2011, pp. 1435–1444.

[46] I. F. Ilyas, T. Rekatsinas, V. Konda, J. Pound, X. Qi, and M. Soliman, "Saga: A platform for continuous construction and serving of knowledge at scale," in *Proceedings of the 2022 International Conference on Management of Data*, 2022, pp. 2259–2272.

[47] "Apache Jena," https://jena.apache.org/, accessed: November 27, 2023.

[48] A. Keely, "SHACLGEN," https://pypi.org/project/shaclgen/, 2023, accessed 17th January, 2023.

[49] K. Kellou-Menouer, N. Kardoulakis, G. Troullinou, Z. Kedad, D. Plexousakis, and H. Kondylakis, "A survey on semantic schema discovery," *VLDB J.*, vol. 31, no. 4, pp. 675–710, 2022. [Online]. Available: https://doi.org/10.1007/s00778-021-00717-x

[50] H. Knublauch, J. A. Hendler, and K. Idehen, "Spin-overview and motivation," *W3C Member Submission*, vol. 22, p. W3C, 2011.

[51] H. Knublauch and D. Kontokostas, "Shapes constraint language (shacl)," *W3C Candidate Recommendation*, vol. 11, no. 8, 2017.

[52] H. Lbath, A. Bonifati, and R. Harmer, "Schema inference for property graphs," in *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*. Nicosia, Cyprus: OpenProceedings.org, 2021, pp. 499–504. [Online]. Available: https://doi.org/10.5441/002/edbt.2021.58

[53] S. Malyshev, M. Krötzsch, L. González, J. Gonsior, and A. Bielefeldt, "Getting the most out of wikidata: semantic technology usage in wikipedia's knowledge graph," in *The Semantic Web–ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part II 17*. Springer, 2018, pp. 376–394.

[54] J. P. McCrae, A. Abele, P. Buitelaar, R. Cyganiak, A. Jentzsch, V. Andryushechkin, and J. Debattista, "The linked open data cloud." [Online]. Available: https://lod-cloud.net/

[55] M. Meimaris, G. Papastefanatos, N. Mamoulis, and I. Anagnostopoulos, "Extended characteristic sets: graph indexing for sparql query optimization," *ICDE*, pp. 497–508, 2017.

[56] N. Mihindukulasooriya, M. R. A. Rashid, G. Rizzo, R. García-Castro, Ó. Corcho, and M. Torchiano, "RDF shape induction using knowledge base profiling," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC*. Pau, France: ACM, 2018, pp. 1952–1959.

[57] C. S. B. (MONICA.), *DATA AND INFORMATION QUALITY: Dimensions, Principles and Techniques*. SPRINGER, 2018.

[58] B. Motik, I. Horrocks, and U. Sattler, "Adding integrity constraints to OWL," in *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*, ser. CEUR Workshop Proceedings, vol. 258. Austria: CEUR-WS.org, 2007.

[59] ——, "Bridging the gap between owl and relational databases," *Journal of Web Semantics*, vol. 7, no. 2, pp. 74–89, 2009.

[60] Neo4j, "neosemantics (n10s): Neo4j rdf & semantics toolkit," https://neo4j.com/labs/neosemantics/, 2023, accessed: August 1, 2023.

[61] T. Neumann and G. Moerkotte, "Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins," in *ICDE*. IEEE, 2011, pp. 984–994.

[62] T. Neumann and G. Weikum, "Rdf-3x: a risc-style engine for rdf," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 647–659, 2008.

[63] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor, "Industry-scale knowledge graphs: Lessons and challenges: Five diverse technology companies show how it's done," *Queue*, vol. 17, no. 2, pp. 48–75, 2019.

[64] P. G. Omran, K. Taylor, S. J. R. Méndez, and A. Haller, "Towards SHACL learning from knowledge graphs," in *Proceedings of the ISWC 2020 Demos and Industry Tracks*, ser. CEUR Workshop Proceedings, vol. 2721. Globally online: CEUR-WS.org, 2020, pp. 94–99.

[65] H. J. Pandit, D. O'Sullivan, and D. Lewis, "Using ontology design patterns to define SHACL shapes," in *Proceedings of the 9th Workshop on Ontology Design and Patterns*, ser. CEUR Workshop Proceedings, vol. 2195. Monterey, USA: CEUR-WS.org, 2018, pp. 67–71.

[66] M. P. Papazoglou, S. Spaccapietra, and Z. Tari, "Database integration: The key to data interoperability," *MIT Press*, 2000.

[67] J. Park and S. Ram, "Information systems interoperability: What lies beneath?" *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 4, pp. 595–632, 2004.

[68] Y. Park, S. Ko, S. S. Bhowmick, K. Kim, K. Hong, and W.-S. Han, "G-CARE: A Framework for Performance Benchmarking of Cardinality Estimation Techniques for Subgraph Matching," in *ACM SIGMOD*, 2020, pp. 1099–1114.

[69] H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods," *Semantic web*, vol. 8, no. 3, pp. 489–508, 2017.

[70] M.-D. Pham, L. Passing, O. Erling, and P. Boncz, "Deriving an emergent relational schema from rdf data," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 864–874.

[71] A. Polleres, R. Pernisch, A. Bonifati, D. Dell'Aglio, D. Dobriy, S. Dumbrava, L. Etcheverry, N. Ferranti, K. Hose, E. Jiménez-Ruiz *et al.*, "How does knowledge evolve in open knowledge graphs?" *Transactions on Graph Data and Knowledge*, vol. 1, no. 1, pp. 11–1, 2023.

[72] E. Prud'hommeaux, J. E. L. Gayo, and H. R. Solbrig, "Shape expressions: an RDF validation and transformation language," in *Proceedings of the 10th International Conference on Semantic Systems, SEMANTiCS 2014, Leipzig, Germany, September 4-5, 2014*. Leipzig, Germany: ACM, 2014, pp. 32–40. [Online]. Available: https://doi.org/10.1145/2660517.2660523

[73] ——, "Shape expressions: an RDF validation and transformation language," in *Proceedings of the 10th International Conference on Semantic Systems, SEMANTiCS*. Leipzig, Germany: ACM, 2014, pp. 32–40.

[74] "PySHACL," https://pypi.org/project/pyshacl/, accessed: November 27, 2023.

[75] K. Rabbani, M. Lissandrini, A. Bonifati, and K. Hose, "Lossless transformations of knowledge graphs to property graphs using standardized schemas," April 2024, unpublished manuscript.

[76] K. Rabbani, M. Lissandrini, and K. Hose, "Optimizing SPARQL queries using shape statistics," in *Proceedings of the 24th International Conference on Extending Database Technology, EDBT*. Nicosia, Cyprus: OpenProceedings.org, 2021, pp. 505–510.

[77] ——, "Shacl and shex in the wild: A community survey on validating shapes generation and adoption," in *Proceedings of the ACM Web Conference 2022*. Online, Lyon, France: ACM, 2022, pp. 260–263. [Online]. Available: https://www2022.thewebconf.org/PaperFiles/65.pdf

[78] ——, "SHACL shapes for DBpedia, LUBM, YAGO-4, and WikiData published on Zenodo." https://doi.org/10.5281/zenodo.5958985, 2022.

[79] ——, "Extraction of validating shapes from very large knowledge graphs," *Proceedings of the VLDB Endowment*, vol. 16, no. 5, pp. 1023–1032, 2023.

[80] ——, "SHACTOR: improving the quality of large-scale knowledge graphs with validating shapes," in *Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023*, S. Das, I. Pandis, K. S. Candan, and S. Amer-Yahia, Eds. ACM, 2023, pp. 151–154. [Online]. Available: https://doi.org/10.1145/3555041.3589723

[81] "RDF4J," https://rdf4j.org/, accessed: November 27, 2023.

[82] T. Sagi, M. Lissandrini, T. B. Pedersen, and K. Hose, "A design space for rdf data representations," *The VLDB Journal*, vol. 31, no. 2, pp. 347–373, 2022.

[83] M. Saleem, M. I. Ali, A. Hogan, Q. Mehmood, and A.-C. N. Ngomo, "Lsq: the linked sparql queries dataset," in *The Semantic Web-ISWC 2015: 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II 14*. Springer, 2015, pp. 261–269.

[84] R. Schaffenrath, D. Proksch, M. Kopp, I. Albasini, O. Panasiuk, and A. Fensel, "Benchmark for performance evaluation of shacl implementations in graph databases," in *Rules and Reasoning: 4th International Joint Conference, RuleML+ RR 2020, Oslo, Norway, June 29–July 1, 2020, Proceedings 4*. Springer, 2020, pp. 82–96.

[85] J. Sequeda and O. Lassila, *Designing and building enterprise knowledge graphs*. Springer Nature, 2022.

[86] J. F. Sequeda, M. Arenas, and D. P. Miranker, "On directly mapping relational databases to rdf and owl," in *Proceedings of the 21st international conference on World Wide Web*, 2012, pp. 649–658.

[87] A. P. Sheth, "Changing focus on interoperability in information systems: from system, syntax, structure to semantics," in *Interoperating geographic information systems*. Springer, 1999, pp. 5–29.

[88] B. Spahiu, A. Maurino, and M. Palmonari, "Towards improving the quality of knowledge graphs with data-driven ontology patterns and SHACL," in *Emerging Topics in Semantic Technologies - ISWC 2018 Satellite Events*, ser. Studies on the Semantic Web, vol. 36. Satellite, USA: IOS Press, 2018, pp. 103–117.

[89] "Stardog," https://www.stardog.com/, accessed: November 27, 2023.

[90] G. Stefanoni, B. Motik, and E. V. Kostylev, "Estimating the cardinality of conjunctive queries over RDF data using graph summarisation," in *WWW*. ACM, 2018, pp. 1043–1052.

[91] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds, "Sparql basic graph pattern optimization using selectivity estimation," in *WWW*, 2008, pp. 595–604.

[92] T. P. Tanon, G. Weikum, and F. M. Suchanek, "YAGO 4: A reason-able knowledge base," in *The Semantic Web - 17th International Conference, ESWC*, ser. Lecture Notes in Computer Science, vol. 12123. Heraklion, Crete, Greece: Springer, 2020, pp. 583–596.

[93] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness, "Integrity constraints in OWL," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI*. Atlanta, Georgia, USA: AAAI Press, 2010.

[94] K. Thornton, H. Solbrig, G. S. Stupp, J. E. L. Gayo, D. Mietchen, E. Prud'Hommeaux, and A. Waagmeester, "Using shape expressions (shex) to share rdf data models and to guide curation with rigorous validation," in *ESWC*. Cham: Springer, 2019, pp. 606–620.

[95] Top Quadrant, "TopBraid," https://www.topquadrant.com/products/topbraid-composer/, 2023, accessed 17th January, 2023.

[96] D. Vrandecic and M. Krötzsch, "Wikidata: a free collaborative knowledgebase," *Commun. ACM*, vol. 57, no. 10, pp. 78–85, 2014.

[97] D. Vrandečić and M. Krötzsch, "Wikidata: a free collaborative knowledgebase," *Communications of the ACM*, vol. 57, no. 10, pp. 78–85, 2014.

[98] W3C, "W3C Data Shapes Test Suite," https://w3c.github.io/data-shapes/data-shapes-test-suite/, accessed: November 27, 2023.

[99] WikiData, "WikiData-2015," https://archive.org/details/wikidata-json-20150518, 2023, accessed 17th January, 2023.

[100] World Wide Web Consortium, "RDFShape," http://rdfshape.weso.es, 2022, accessed 20th January.

[101] World Wide Web Consortium *et al.*, "Owl 2 web ontology language document overview," *World Wide Web Consortium*, 2012.

[102] R. Zhang, P. Liu, X. Guo, S. Li, and X. Wang, "A unified relational storage scheme for rdf and property graphs," in *Web Information Systems and Applications: 16th International Conference, WISA 2019, Qingdao, China, September 20-22, 2019, Proceedings 16*. Springer, 2019, pp. 418–429.

Part II

Papers

Paper A

SHACL and ShEx in the Wild: A Community Survey on Validating Shapes Generation and Adoption

Kashif Rabbani, Matteo Lissandrini, Katja Hose

The paper has been published in the
In Companion Proceedings of The Web Conference 2022, Virtual Event / Lyon, France, April 25 - 29, 2022. DOI: 10.1145/3487553.3524253

Abstract

Knowledge Graphs (KGs) are widely used to represent heterogeneous domain knowledge on the Web and within organizations. Various methods exist to manage KGs and ensure the quality of their data. Among these, the Shapes Constraint Language (SHACL) and the Shapes Expression Language (ShEx) are the two state-of-the-art languages to define validating shapes for KGs. Since the usage of these constraint languages has recently increased, new needs arose. One such need is to enable the efficient generation of these shapes. Yet, since these languages are relatively new, we witness a lack of understanding of how they are effectively employed for existing KGs. Therefore, in this work, we answer How validating shapes are being generated and adopted? *Our contribution is threefold. First, we conducted a community survey to analyze the needs of users (both from industry and academia) generating validating shapes. Then, we cross-referenced our results with an extensive survey of the existing tools and their features. Finally, we investigated how existing automatic shape extraction approaches work in practice on real, large KGs.* Our analysis shows the need for developing semi-automatic methods that can help users generate shapes from large KGs.

Rabbani, K., Lissandrini, M., & Hose, K. (2022, April). SHACL and ShEx in the wild: a community survey on validating shapes generation and adoption. In *Companion Proceedings of the Web Conference 2022 (pp. 260-263).*
https://doi.org/10.1145/3487553.3524253
The layout has been revised.

1 Introduction

The popularity of Knowledge Graphs (KGs) has consistently grown in the last decade due to the advent of public KGs, such as DBpedia [1], YAGO [24], and WikiData [27] and their adoption among companies [14]. These KGs (represented in RDF) are massive, diverse, and most importantly incomplete due to the evolving nature of human knowledge. Given the importance of KGs, it is paramount to ensure the quality of the information they represent. Validating languages, i.e., the shapes constraints languages (SHACL [11] and ShEx [17]) are used to ensure the data quality in KGs by defining integrity constraints in the form of shapes, i.e., high-level structural constraints for entities within a KG. While SHACL and ShEx differ at a syntactic level, they both allow to specify a set of characterizing properties and attributes for classes of entities that are expected to be hold within a KG [19]. For instance, they both allow to specify that all entities of type "Student" need to have "name" and "registration number" as properties, and be linked to at least one "Course". Shapes also specify data types for these properties, e.g., *String*, *Integer*, or *IRI*. Shapes can also be used for purposes other than validation, such as to design user interfaces [6, 29], or optimize query processing [19].

There exist various tools to help define validating shapes, either manually or semi-automatically, such as Astrea [3], TopBraid Composer [26], the SHACLGEN [8] python library, ShapeDesigner [2] and SheXer [5] to semi-automatically generate SHACL and ShEx from ontologies and KGs data, as well as approaches to define shapes using profiling [13] and ontology design patterns [16]. *Naturally, when a KG contains hundreds of classes, each having many attributes, manually specifying (post-hoc) the necessary shapes becomes a tedious and unmanageable task.* On the other hand, the approaches helping automatic generation of validating shapes produce a large number of shape constraints (for nodes and properties) such that it becomes non-trivial to verify the validity of the generated constraints.

To better understand the needs of users, both in industry and academia, to generate shapes for large KGs, we conducted an online community survey with a set of questions to learn how (and up to what extent) they generate and use shapes. We used Google Forms to create an online survey[1] and shared it with researcher and practitioners across different communities, e.g., within members of the W3C mailing list[2], the Solid project[3], the BlueBrainNexus[4] project, the Bayer-Group COLID team[5], and Slack channels[6] for Knowledge

[1] https://forms.gle/93KFZH5vcGy7Et27A
[2] https://lists.w3.org/Archives/Public/semantic-web/
[3] https://gitter.im/solid/
[4] https://bluebrainnexus.io
[5] https://github.com/Bayer-Group/COLID-Documentation
[6] https://knowledgegraphconf.slack.com, https://iswc-conf.slack.com

Graph and ISWC conferences. Additionally, we have contacted other related Semantic Web communities developing various tools and approaches to deal with validating shapes. When we asked users to participate in our survey, they all manifested great interest in the results of this study. Hence, we promised to re-share our findings with the community. This work is also a way to keep up with that promise. Therefore, in the following *we first report our findings by analyzing the results obtained from our online community survey*, then *we discuss the state of the art in validating shapes*. Finally, *we report on some experimental results in actually using such tools*. Our analysis highlights a number of *gaps when it comes to generating shapes automatically* to validate existing (possibly noisy or erroneous) KGs, thus we discuss a number of future research directions based on our findings.

2 Community Survey

This section summarizes the questions and answers to our survey. The survey contained nine questions and received in total 30 answers. The survey was conducted online, and the results were collected anonymously between November 2021 and January 2022. Among the questions, we surveyed also the area of occupation of the respondents, i.e., if they belong to Academia, Industry, or both. Answers to this question showed that 53% of the respondents are from Industry, 27% from Academia, and 20% from both (see Figure A.2a). In the following, where relevant, we will report for each statistics, in parenthesis, the split of answers by respondents from academia, industry, or both respectively.

Our main question asked *"how the validating shapes were being generated"* in general. We provided three answer options and allowed the respondents to also add a free text answer. Answers to this question are presented in Figure A.1, where numbers in circles represent the number of participants. The results show that most of the respondents (i.e., 26 - split 6/14/6) generate shapes manually, while 13 respondents (split 5/5/3) generate shapes from existing ontologies and 7 respondents (split 1/4/2) generate shapes from RDF graphs instance data, while only 1 respondent declared to use "RDF forms". The results overlap as it was a multi-choice question where respondents had chosen more than one option.

We then asked *which tools or methods the respondents used in practice* and provided a list of state-of-the-art tools and approaches that can help in generating shapes (see Table A.1) along with a free text answer option as well. To our surprise, respondents use a very heterogeneous set of tools and methods, with the most used being TopBraid Composer [26] used by 33% (10) of respondents (split 1/7/2), Protégé was used by 20% (6) respondents (split 3/3/0), RDFShape [28] by another 16.7% (5) respondents (split 2/1/2),

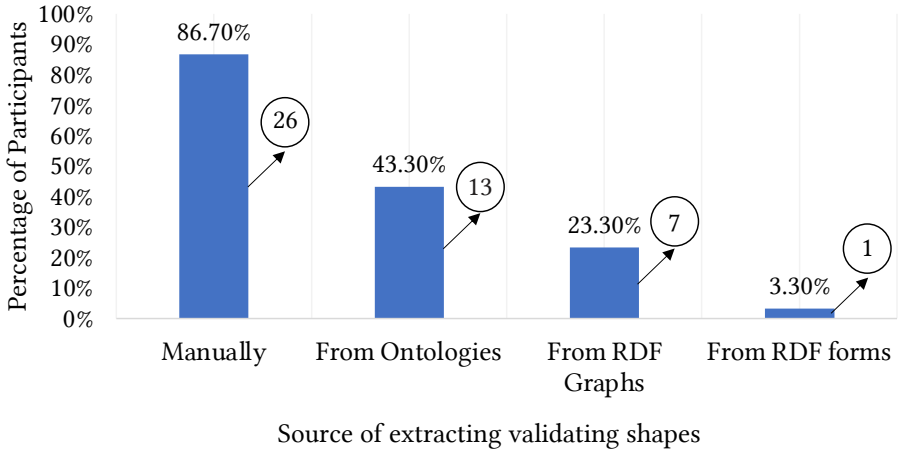


Fig. A.1: Analysis on extraction of validating shapes

SheXer [5] was used by 10% (3) respondents (split 0/1/2), and text editors are used by 16.7% (5) respondents (split 2/4/0). The rest of the tools are used by 1 or 2 respondents on average. In addition to that, one of the respondents mentioned that they generate shapes by applying custom rules via Python scripts, another by using tabular formats (like Excel) to curate shapes manually. Lastly, one respondent commented: "I am looking for a suitable tool".

We further asked a number of questions related to the characteristics of the graphs for which the respondents were generating shapes. Results showed that 38% of respondents generate shapes from RDF graphs containing up to 100K triples (split 4/5/1), 17% use graphs having 100K to 1 million triples (split 0/5/0), and 45% use graphs containing more than 1 million triples (split 4/6/3), see Figure A.2a and A.2b. The great majority 48% (14) of KGs are described by ontologies containing between 10 and 1000 classes and 10-50 distinct properties (see Figure A.2c and A.2d). While 32% (9) of the respondents generate up to 10 shapes, 47% (13) generate between 10-100 shapes, and 14% (4) generate more than 100 shapes and in some cases even more than 1000 shapes (see Figure A.2e).

In addition to that, 55% (16) of the respondents generate shapes for the entire graph, and 43% (13) of them generate shapes for specific portions of the graph (see Figure A.2f) due to various reasons (answers to our last question). These reasons include (i) use case specific requirements to focus on shapes targeting only the important classes of the graph (10%), (ii) interest or requirement to get a specific maintained view of the data graph or subset (30%), (ii) limitation of the known business rules to apply only to part of the graph (10%), (iv) the scalability of existing methods to generate shapes is

insufficient for very large graphs (10%), (v) the decision to generate shapes progressively (such as for WikiData) to accommodate evolution through time and change in requirements (30%), and (vi) the requirement of validating only non-logical statements that cannot be validated using (already existing) OWL constraints (15%).

The results show that *most users work with fairly large graphs with tens or hundreds of classes and thousands or millions of triples*. Yet, *most users still generate shapes manually*; this way, they can generate only a handful of shapes (not enough to cover the entire graph). Thus, there is also the need for tools in supporting the scalable generation of shapes for very large graphs, which should allow to extract shapes for only a specific portion of a KG. Therefore, we conducted an extensive analysis of tools used to generate validating shapes and present our findings in the next section to better understand the capabilities offered by existing approaches when it comes to helping users generate shapes more efficiently.

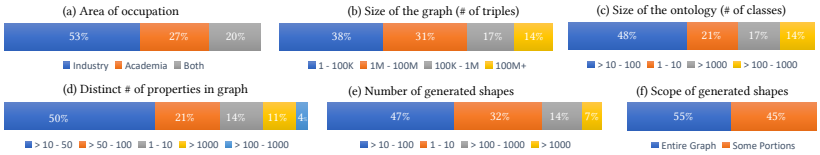


Fig. A.2: Survey Analysis: Answers to statistical questions

3 State of the art

Integrity constraints over KGs were initially defined using OWL [25]. Later on, the SPARQL Inferencing Notation (SPIN) [10], a SPARQL-based rule and constraint language was introduced to enforce constraints over KGs using SPARQL queries. SHACL [11] is known as the next generation of SPIN and has become a W3C recommended language in 2017. Similar to SHACL, ShEx [18] is a constraint language built on regular bag expressions inspired by schema languages for XML. Validating schemas have been adopted for type and compliance checking [12, 20, 21], as well as purposes other than validation as well. Specifically, for optimizing query processing in KGs [19], building and automatically generating forms to populate RDF datasets [6, 29], intelligent geoprocessing [7], and automatic detection of metadata errors in clinical studies [9].

Shape Extraction: There has been recent development to derive validating shapes from existing KG instance data; various applications are created to assist the process of shapes extraction, which can potentially be divided

into two phases, i.e., preprocessing and shapes construction. The *preprocessing* phase involves collecting information from the input dataset about classes and their properties, which are then used in the *shapes construction* phase. The preprocessing phase can be conducted in two alternative ways: query-based and non-query-based. The query-based solutions involve loading the KG into a triplestore able to answer SPARQL queries. The triplestore is then used to access the information required for the shapes construction phase. SHACLGEN [8], RDFShape [28], and ShapeDesigner [2] are query-based solutions to generate SHACL or ShEx shapes from existing RDF data. Non-query-based solutions, instead, parse RDF data stored as files on disk. SheXer [5] is the only tool to generate ShEx shapes that supports both a triplestore and RDF files as input. Spahiu et al. [22] proposed a solution based on a data profiling tool called ABSTAT [23] to generate semantic profiles and transform them into SHACL to improve the quality of the KGs. Mihindukulasooriya et al. [13] proposed a KG data profiling-based RDF shape induction approach by using predictive modeling to generate shapes. SHACLearner [15] is a method to learn SHACL constraints based on Inverse Open Path rules (IOP) rules. Astrea [3] is an ontology-based approach to extract SHACL from ontologies. Finally, RML2SHACL [4] also generates SHACL shapes but it requires RML mappings. We have classified existing approaches in Table A.1 based on their features (expanded from a previous short survey [3]), i.e., support for shapes extraction from data or ontologies, support for automatic extraction of shapes, support for shapes extraction from a SPARQL triplestore, and whether they extract SHACL, ShEx, or both types of validating shapes. *Overall, as we show later, we note that shapes extracted by these methods are often incomplete, i.e., they do not implement functionalities to generate all types of shape constraints*, e.g., they often do not produce sh:class and cardinality constraints.

Table A.1: State-of-the-art to extract validating shapes

Approach	*Extracted from*		*Auto-matic*	*Triple-store*	*Type*
	data	*ontology*			
Shape Induction [13]	✔	✗	✔	✔	SHACL,ShEx
SheXer [5]	✔	✗	✔	✔	SHACL,ShEx
Spahiu et al. [22]	✔	✗	✔	✔	SHACL
ShapeDesigner. [2]	✔	✗	✔	✔	SHACL,ShEx
SHACLGEN [8]	✔	✔	✔	✔	SHACL
TopBraid [26]	✔	✔	✔	✔	SHACL
Pandit et al. [16]	✗	✔	✗	✔	SHACL
Astrea [3]	✗	✔	✔	✗	SHACL
SHACLearner [15]	✔	✗	✔	✗	SHACL

4 Limitations and Opportunities

In light of the results presented above, this section highlights the existing gaps between the capabilities of current tools and the real user needs. After this analysis, we present a set of important research directions for automatic shape extraction.

Among the existing approaches, some tools support extraction of validating shapes from ontologies only (such as Astrea [3]), others support extraction from instance data, and very few support automatic generation of validating shapes from both ontology and instance data (see Table A.1). Methods that generate shapes from ontologies assume the provided ontology to be complete (i.e., providing complete coverage of the instances in the KG and their properties) and of small size (they do not expect more than a few hundred classes). Approaches that extract shapes from RDF instance data assume that the KG is particularly small (in terms of the number of instances and classes) or already available in a triplestore. When this is not the case, they load it into an in-memory triplestore, which is problematic for large graphs, as further discussed below. We believe these assumptions clash with a number of real use cases. As a matter of fact, in the results of our survey, we see that, despite the existence of tools and approaches to extract validating shapes automatically, *most users are still generating shapes manually*. To understand this better, we ran some experiments to find out the real capabilities *of existing tools for automatically extracting shapes from RDF graphs.* The experiments are performed using state-of-the-art shapes extraction tools and approaches such as SheXer [5], ShapeDesigner [2], and SHACLGEN [8]. All experiments are performed on a machine with 24 cores and 256GB of RAM using a 2021 dump of DBpedia [1], YAGO-4 [24], and a version of LUBM scaled to 91M triples. More details on our experiments and datasets are available online[7]. We tried similar experiments for WikiData [27] as well, but the initial results show that existing methods are not able to handle its scale (they exhausted all memory or did not manage to produce an output in several days). *Therefore, we highlight that more research is needed to design scalable methods to extract validating shapes from large KGs.*

Our first finding from these experiments is that some tools are capable of handling only relatively small KGs. In particular ShapeDesigner [2] crashed with datasets with a few millions triples, while SHACLGEN [8] is not suited to extract shapes of datasets having hundreds of classes as it required days to generate shapes for YAGO-4 (8,897 classes). Furthermore, we note that both ShapeDesigner [2] and SHACLGEN [8] load the whole graph into a triplestore to generate shapes. Yet, extracting shapes by querying a triplestore is

[7] https://relweb.cs.aau.dk/validatingshapes/
Zenodo: https://doi.org/10.5281/zenodo.5958985

particularly inefficient when extracting shapes for the entire graph. Compared to parsing and analyzing a file, which can exploit optimized batch processing over multiple scans of the data, extracting shapes when the KG is in a triplestore requires to run queries for each target class to extract all necessary information. When processing RDF files directly, SheXer [5] provides better performance and was able to extract shapes from DBpedia consuming a maximum of 18GB of RAM in 26 minutes (LUBM: 33GB RAM in 58 minutes, YAGO-4: 24GB RAM in 117 minutes).

Our second finding relates to the usefulness and reliability of the shapes extracted from the instance data. Shapes are useful only if they are complete (containing all the required constraints to validate the input graph) and reliable if they do not contain constraints representing spurious data. Our analysis (via manual inspection) of the shapes extracted by these tools revealed that *none of the approaches extracts all the required constraints* for property shapes; for instance, we did not find any constraint for non-literal predicate objects (e.g., to indicate that objects for "takes course" should be of type "Course"). Furthermore, we saw that some property shapes were extracted because of the presence of some nodes with spurious or erroneous predicates (e.g., a city mistaken as a member of a musical band). To further investigate these issues, we implemented a Java application[7] to extract also the missing shape constraints and report their support in the graph, e.g., how many entities were satisfying a specific constraint. *We found that generating automatically the complete set of shapes would produce hundreds or thousands of node and property shapes* each having multiple constraints. Yet, since the source KGs are naturally noisy and incomplete, *those shapes and constraints are often unreliable.* In particular, for DBpedia, we extracted 426 node shapes and 11,916 property shapes, which have 38,454 non-literal and 5,335 literal constraints. Similarly, for YAGO-4, we extracted 8,897 node shapes and 76,765 property shapes, with a total of 315,413 non-literal and 50,708 literal constraints. We have published the extracted SHACL shapes on Zenodo[7]. Given the above numbers of extracted constraints for DBpedia and YAGO-4, it is non-trivial to manually establish the validity and the usefulness of thousands of automatically generated shape constraints. The only existing approach in this direction is SheXer [5], which supports filtering of shapes based on a "trustworthiness" score (even though we found this score does not directly translate into how frequently a shape is satisfied in a dataset, and thus it is hard to tune).

In conclusion, referring to the results of our community survey (Figure A.1), we see a connection between the limitations of existing shapes extraction approaches in effectively supporting the real user needs and the (for now) common practice of generating shapes manually. Since validating shapes have become an essential tool to ensure the quality and completeness of large KGs in many organizations, these results suggest that *more research is needed to help users generate useful validating shapes for existing large KGs*, consid-

ering both the scalability of the approach and the informativeness and utility of the extracted shapes. This will help users curate and maintain high-quality large-scale KGs.

References

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *The semantic web*, 2007, pp. 722–735.

[2] I. Boneva, J. Dusart, D. F. Alvarez, and J. E. L. Gayo, "Shape designer for shex and shacl constraints," in *ISWC 2019*, 2019.

[3] A. Cimmino, A. Fernández-Izquierdo, and R. García-Castro, "Astrea: Automatic generation of shacl shapes from ontologies," in *ESWC*, 2020, pp. 497–513.

[4] T. Delva, B. De Smedt, S. Min Oo, D. Van Assche, S. Lieber, and A. Dimou, "Rml2shacl: Rdf generation taking shape," in *Proceedings of the 11th on Knowledge Capture Conference*, 2021, pp. 153–160.

[5] D. Fernandez-Álvarez, J. E. Labra-Gayo, and D. Gayo-Avello, "Automatic extraction of shapes using shexer," *Knowledge-Based Systems*, vol. 238, p. 107975, 2022.

[6] A. Hogan, "Book," in *The Web of Data*. Cham: Springer, 2020.

[7] Z.-W. Hou, C.-Z. Qin, A. Zhu, Y.-J. Wang, P. Liang, Y.-J. Wang, Y.-Q. Zhu *et al.*, "Formalizing parameter constraints to support intelligent geoprocessing: A shacl-based method," *ISPRS International Journal of Geo-Information*, vol. 10, no. 9, p. 605, 2021.

[8] A. Keely, "SHACLGEN," https://pypi.org/project/shaclgen/, 2023, accessed 17th January, 2023.

[9] D. Keuchel and N. Spicher, "Automatic detection of metadata errors in a registry of clinical studies using shapes constraint language (shacl) graphs," *Studies in health technology and informatics*, vol. 281, pp. 372–376, 2021.

[10] H. Knublauch, J. A. Hendler, and K. Idehen, "Spin-overview and motivation," *W3C Member Submission*, vol. 22, p. W3C, 2011.

[11] H. Knublauch and D. Kontokostas, "Shapes constraint language (shacl)," *W3C Candidate Recommendation*, vol. 11, no. 8, 2017.

[12] M. Leinberger, P. Seifer, C. Schon, R. Lämmel, and S. Staab, "Type checking program code using shacl," in *ISWC*, 2019, pp. 399–417.

[13] N. Mihindukulasooriya, M. R. A. Rashid, G. Rizzo, R. García-Castro, O. Corcho, and M. Torchiano, "Rdf shape induction using knowledge base profiling," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 2018, pp. 1952–1959.

[14] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor, "Industry-scale knowledge graphs: lessons and challenges," *Communications of the ACM*, vol. 62, no. 8, pp. 36–43, 2019.

[15] P. G. Omran, K. Taylor, S. J. R. Méndez, and A. Haller, "Towards shacl learning from knowledge graphs." in *ISWC (Demos/Industry)*, 2020, pp. 94–99.

[16] H. J. Pandit, D. O'Sullivan, and D. Lewis, "Using Ontology Design Patterns To Define SHACL Shapes." in *WOP@ ISWC*, 2018, pp. 67–71.

[17] E. Prud'hommeaux, J. E. L. Gayo, and H. Solbrig, "Shape expressions: an RDF validation and transformation language," in *ICSS*, 2014, pp. 32–40.

[18] E. Prud'hommeaux, J. E. Labra Gayo, and H. Solbrig, "Shape expressions: an rdf validation and transformation language," in *Proceedings of the 10th International Conference on Semantic Systems*, 2014, pp. 32–40.

[19] K. Rabbani, M. Lissandrini, and K. Hose, "Optimizing sparql queries using shape statistics," in *Proceedings of the 24th International Conference on Extending Database Technology, EDBT*, 2021.

[20] L. Robaldo, "Towards compliance checking in reified i/o logic via shacl," in *International Conference on Artificial Intelligence and Law*, 2021, pp. 215–219.

[21] U. Şimşek, K. Angele, E. Kärle, O. Panasiuk, and D. Fensel, "Domain-specific customization of schema. org based on shacl," in *ISWC*, 2020, pp. 585–600.

[22] B. Spahiu, A. Maurino, and M. Palmonari, "Towards improving the quality of knowledge graphs with data-driven ontology patterns and shacl." in *ISWC (Best Workshop Papers)*, 2018, pp. 103–117.

[23] B. Spahiu, R. Porrini, M. Palmonari, A. Rula, and A. Maurino, "Abstat: ontology-driven linked data summaries with pattern minimalization," in *European Semantic Web Conference*. Springer, 2016, pp. 381–395.

[24] T. P. Tanon, G. Weikum, and F. Suchanek, "Yago 4: A reason-able knowledge base," in *ESWC*, 2020, pp. 583–596.

[25] J. Tao, E. Sirin, J. Bao, and D. McGuinness, "Integrity constraints in owl," in *AAAI*, vol. 24, no. 1, 2010.

[26] Top Quadrant, "TopBraid," https://www.topquadrant.com/products/topbraid-composer/, 2023, accessed 17th January, 2023.

[27] D. Vrandečić and M. Krötzsch, "Wikidata: a free collaborative knowledgebase," *Communications of the ACM*, vol. 57, no. 10, pp. 78–85, 2014.

[28] World Wide Web Consortium, "RDFShape," http://rdfshape.weso.es, 2022, accessed 20th January.

[29] J. Wright, S. J. R. Méndez, A. Haller, K. Taylor, and P. G. Omran, "Schimatos: a shacl-based web-form generator for knowledge graph editing," in *ISWC*, 2020, pp. 65–80.

Paper B

Extraction of Validating Shapes from very large Knowledge Graphs

Kashif Rabbani, Matteo Lissandrini, Katja Hose

The paper has been published in the
In Proceedings of the Very Large Databases 2023 (Volume 16), August 28 - Sept 02, 2023, Vancouver, Canada.. DOI: 10.14778/3579075.3579078

Abstract

Knowledge Graphs (KGs) represent heterogeneous domain knowledge on the Web and within organizations. There exist shapes constraint languages to define validating shapes *to ensure the quality of the data in KGs. Existing techniques to extract validating shapes often fail to extract complete shapes, are not scalable, and are prone to produce spurious shapes. To address these shortcomings, we propose the* Quality Shapes Extraction *(QSE) approach to extract validating shapes in very large graphs, for which we devise both an exact and an approximate solution. QSE provides information about the reliability of shape constraints by computing their confidence and support within a KG and in doing so allows to identify shapes that are most informative and less likely to be affected by incomplete or incorrect data. To the best of our knowledge, QSE is the first approach to extract a complete set of validating shapes from WikiData. Moreover, QSE provides a 12x reduction in extraction time compared to existing approaches, while managing to filter out up to 93% of the invalid and spurious shapes, resulting in a reduction of up to 2 orders of magnitude in the number of constraints presented to the user, e.g., from 11,916 to 809 on DBpedia.*

Rabbani, K., Lissandrini, M., & Hose, K. (2023). Extraction of validating shapes from very large knowledge graphs. In *Proceedings of the VLDB Endowment, Vol. 16, No. 5 ISSN 2150-8097.*

https://doi.org/10.14778/3579075.3579078

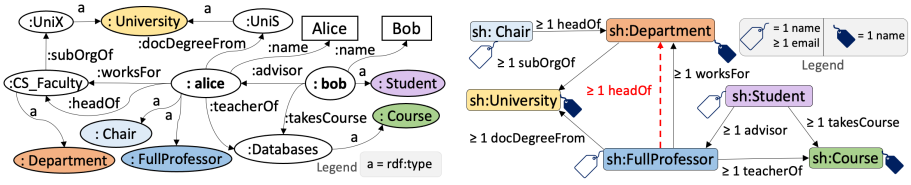
The layout has been revised.

1 Introduction

Knowledge Graphs (KGs), stored as collections of triples of the form $\langle subject, relation, object \rangle$ using the Resource Description Framework (RDF) [10], are in widespread use both within companies [29, 43, 44] and on the Web [46, 49]. Nonetheless, as KGs quickly accrue more data, practical applications impose further demands in terms of quality assessment and validation [34, 38, 52]. Hence, shapes constraint languages, e.g., SHACL [23], and ShEx [35], have been proposed to validate KGs by enforcing constraints represented in the form of *validating shapes*. For instance, we can express that an entity of type Student requires a name, a registration number, and should be enrolled in some courses; and that these attributes should be of type string, integer, and Course, respectively – see Figures B.1a and B.1b for an example KG and corresponding shapes.

Often, validating shapes are manually specified by domain experts. Yet, when trying to specify validating shapes for already-existing large-scale KGs, data scientists are in need of tools that can speed up this process [38]. Thus, various tools have been proposed to automatically [9, 12, 20, 26] or semi-automatically [5, 32, 36] produce a set of validating shapes for a target KG. Unfortunately, these methods suffer from 3 important limitations: (1) they are not able to produce complete shapes, e.g., they can identify that a student should have a property of type takesCourse but they do not extract the fact that the object should be of type Course; (2) the shapes they produce are easily affected by errors and inconsistencies in the KG, e.g., if some departments, by mistake, are attached the property hasAdvisor, a corresponding *spurious shape* is extracted; and (3) they do not scale to large KGs, e.g., they cannot process the full English WikiData, and they take days to process a subset of it. Therefore, in this work, we present the first techniques for *efficient extraction of validating shapes from very large existing KGs that also ensures robustness against the effects of spuriousness*.

Spuriousness poses important challenges to automatic shape extraction methods. For instance, in DBpedia [4], some of the entities representing musical bands are wrongly assigned to the class dbo:City. As a consequence,



(a) RDF Graph **(b)** Validating Shapes

Fig. B.1: An example RDF Graph and Validating Shapes

when shapes are extracted from its instance data using existing approaches, the resulting node shape for dbo:City specifies that cities are allowed optional properties like dbo:genre and dbo:formerBandMember. Hence, due to the effect of *spuriousness*, existing approaches generate tens of thousands of shapes (our experiments show that standard extraction processes produce more than 2 million property shapes for WikiData [49]). Thus, it becomes unmanageable for domain experts to manually identify valid shapes. SheXer [12] is the only existing approach that attempts to tackle this issue by filtering shapes based on a "trustworthiness" score. Unfortunately, this score does not directly translate into how frequently a shape is satisfied in a dataset, so it is still prone to generate spurious shapes, and it is also hard to tune. Furthermore, SheXer is not able to efficiently process large KGs.

Therefore, to tackle the issue of *spuriousness*, we study and formalize the problem of *support-based shapes extraction* and propose the Quality Shapes Extraction (QSE) approach as a solution to this problem. To tackle the issue of *scalability*, we devise two efficient algorithms, QSE-Exact and QSE-Approximate. *Hence,* QSE *can filter out shapes affected by spurious or erroneous data based on robust and easily understandable measures.* QSE allows shapes extraction both from KGs available as files as well as SPARQL endpoints. Moreover, our efficient approximation algorithm enables shape extraction even on a commodity machine by sampling the KG entities via a dynamic multi-tiered reservoir sampling technique.

We perform a thorough experimental evaluation using both synthetic (LUBM [18]) and real (DBpedia [4], YAGO-4 [46], WikiData [49]) KGs demonstrating the benefits of our approach. The shapes produced by our approach are of high quality and instrumental for easily finding errors in real KGs. The results show that QSE-Exact can extract shapes from the entire WikiData's 2015 dump in 16 minutes and from 2021's dump (1.9B triples) in 2.5 hours. Similarly, QSE-Approximate can extract shapes from WikiData's 2021 dump in 90 minutes on a 32GB machine while still achieving 100% precision and 95% recall in the set of shapes produced. Hence, our sampling strategy is accurate and efficient both when extracting shapes from a file as well as when using an endpoint.

2 RDF Shapes and the QSE Problem

In the following, we first introduce the KG data model and the concepts of validating shapes, their support, and confidence, then we define our focus: the Quality Shapes Extraction problem.

2.1 Preliminaries

The standard model for encoding KGs is the Resource Description Framework (RDF [10]), which describes data as a set of $\langle s, p, o\rangle$ triples stating that a subject s is in a relationship with an object o through predicate p. Therefore, we define an RDF graph as follows:

Definition B.1 (RDF graph)
Given pairwise disjoint sets of IRIs $\mathcal{I}$, blank nodes $\mathcal{B}$, and literals $\mathcal{L}$, an RDF Graph $\mathcal{G}{:}\langle N, E\rangle$ is a graph with a finite set of nodes $N{\subset}(\mathcal{I}{\cup}\mathcal{B}{\cup}\mathcal{L})$ and a finite set of edges $E{\subset}\{\langle s, p, o\rangle{\in}(\mathcal{I}{\cup}\mathcal{B}) \times \mathcal{I} \times (\mathcal{I}{\cup}\mathcal{B}{\cup}\mathcal{L})\}$.

Moreover, we distinguish two special subsets of the IRIs $\mathcal{I}$: predicates $\mathcal{P}$ and classes $\mathcal{C}$. The set of predicates $\mathcal{P}{\subset}\mathcal{I}$ is the subset of IRIs that appear in the predicate position p in any $\langle s, p, o\rangle{\in}\mathcal{G}$. Among predicates $\mathcal{P}$, we identify the type predicate $\mathbf{a}{\in}\mathcal{P}$, which corresponds to IRI rdf:type [51] or wdt:P31 WikiData [49], as the predicate that connects all entities that are instances of a class to the node representing the class itself, i.e., their type. Thus, all the IRIs that are classes in $\mathcal{G}$ form the subset $\mathcal{C}{:}\{c{\in}\mathcal{I}|\exists s{\in}\mathcal{I}$ s.t. $\langle s, \mathbf{a}, c\rangle{\in}\mathcal{G}\}$.

Given a KG $\mathcal{G}$, a set of *validating shapes* represents integrity constraints in the form of a shape schema $\mathcal{S}$ over $\mathcal{G}$. Since the shape schema describes shapes associated with node types and their connections to other attributes and node types, we can also visualize the shape schema $\mathcal{S}$ as a particular type of graph (see Figures B.1a and B.1b). Therefore, in the following, we refer to two concepts: the *data graph* $\mathcal{G}$ and the *shape graph* derived from $\mathcal{S}$. The *data graph* is the RDF graph $\mathcal{G}$ to be validated, while the *shape graph* consists of constraints in the form of the shape schema $\mathcal{S}$ against which entities of the data graph are validated. These constraints are defined using node and property shapes. In the following, we adopt the previously defined syntax [42] to refer to the set $\mathcal{S}$ according to the SHACL *core constraint components* [50]. Finally, while validating shapes can also be expressed in ShEx [34], our approach can be trivially extended to output ShEx directly, or it can exploit existing SHACL to ShEx converters [53]. Thus, without loss of generality, we focus on the current standard for SHACL shapes in the following.

Definition B.2 (Shape Schema)
A SHACL shape schema consists of a set of node shapes $\mathcal{S}$, with $\langle s, \tau_s, \Phi_s\rangle{\in}\mathcal{S}$, where s is the shape *name*, $\tau_s{\in}\mathcal{C}$ is the *target class*, and Φ_s is a set of property shapes of the form $\phi_s{:}\langle \tau_\mathsf{p}, \mathsf{T}_\mathsf{p}, \mathsf{C}_\mathsf{p}\rangle$, where $\tau_\mathsf{p}{\in}\mathcal{P}$ is called the target property, $\mathsf{T}_\mathsf{p}{\subset}\mathcal{I}$ contains either an IRI defining a *literal type*, e.g., xsd:string, or a set of IRIs – called *class type constraint*, and C_p is a pair $(n, m) \in \mathbb{N} \times (\mathbb{N}{\cup}\{\infty\})$. $n{\leq}m$ – called min and max *cardinality constraints*.

Therefore, given a node shape $s{\in}\mathcal{S}$ for the *target class* $\tau_s{\in}\mathcal{C}$, Φ_s defines which properties each instance of τ_s can or should be associated with. For instance,

the shape $\langle$sh:Student, :Student, $\{\phi_{s_1}, \phi_{s_2}\}\rangle$ from Figure B.1b, contains a node shape for target class :Student and enforces two property shapes ϕ_1 and ϕ_2. The property shape ϕ_1 has a target property τ_p= :name, a literal type constraint T_p= xsd:string, and the cardinality constraints C_p=(1, 1). Similarly, the property shape ϕ_2 has a target property τ_p=:takesCourse, a class type constraint T_p= :Course, and the cardinality constraint $C_p = (1, \infty)$.

When validating a graph $\mathcal{G}$ against a shape schema $\mathcal{S}$ having a node shape $\langle s, \tau_s, \Phi_s\rangle \in \mathcal{S}$, we verify that each entity $e \in \mathcal{G}$ that is an instance of τ_s satisfies all the constraints Φ_s. Note that we use the term entity and node interchangeably throughout the paper. Thus, we define the semantics of $\mathcal{S}$ as follows:

Definition B.3 (Validating Shape Semantics)
Given a node shape $\langle s, \tau_s, \Phi_s\rangle \in \mathcal{S}$, a graph $\mathcal{G}$, and an entity e s.t. $\langle e, \mathbf{a}, \tau_s\rangle \in \mathcal{G}$, we have that s validates e, and we write $e \models_{\mathcal{G}} \phi$, if for every property shape ϕ_s:$\langle \tau_p, T_p, C_p\rangle \in \Phi_s$ the following conditions hold:

- If T_p is a literal type constraint, then for every triple $(e, \tau_p, l) \in \mathcal{G}$, l is a literal of type T_p.
- If T_p is a set of class type constraints $T_p = \{t_1, t_2, ...t_n\}$, then for every triple $(e, \tau_p, o) \in \mathcal{G}$, it holds that $\forall t \in T_p$, o is an instance of t (or of a subclass of t) and if $\exists S_t \in \mathcal{S}$, $o \models_{\mathcal{G}} S_t$.
- $n \leq |\{(s, p, o) \in \mathcal{G} : s = e \wedge p = \tau_p\}| \leq m$, where $C_p = (n, m)$.

Here we study the case where $\mathcal{G}$ is given, and we want to extract the set of validating shapes $\mathcal{S}$ that validates every class in $\mathcal{C}$ from $\mathcal{G}$. This is the *shapes extraction* problem. In this case, existing automatic approaches [38] assume the graph to be correct, then iterate over all entities in it, and extract for each entity e all necessary shapes that validate e. The union of all such shapes is assumed to be the final schema $\mathcal{S}$. This is useful when we want to validate new data that will be added *in the future* to the KG so that it will conform to the data already in the graph. Unfortunately, this approach will produce spurious shapes. For instance, in Figure B.1, since :alice has both type Full Professor and Chair, when parsing the triple (:alice, :headOf, :CS_Faculty), the property shape headOf (the red dotted arrow in Figure B.1b) is assigned to both node shapes, instead of assigning it to the Chair node shape only.

2.2 Shapes Support and Confidence

To contrast the effect of spuriousness, we want to exploit statistics on how often properties are applied to entities of a given type. Therefore, we introduce the notion of *support* and *confidence* for shape constraints to study the reliability of extracted shapes. These concepts are inspired by the well-known theory developed for the task of frequent patterns mining [19] and the concept of MNI support for graph patterns [7]. The MNI support of a

graph pattern is the minimum cardinality of the *set of all nodes of $\mathcal{G}$ that are mapped to a specific pattern node by some isomorphism* across all the nodes of the pattern. In our approach, a property shape corresponds to a node- and edge-labeled graph pattern. Thus, given the shape $s{:}\langle s, \tau_s, \Phi_s\rangle \in \mathcal{S}$ its support is the number of entities that are of type τ_s, while the support of a property shape $\phi_s{:}\langle \tau_\mathsf{p}, \mathsf{T}_\mathsf{p}, \mathsf{C}_\mathsf{p}\rangle \in \Phi_s$ is the cardinality of entities conforming to it.

Definition B.4 (Support of ϕ_s)
Given a shape $\langle s, \tau_s, \Phi_s\rangle \in \mathcal{S}$ with shape constraint $\phi_s{:}\langle \tau_\mathsf{p}, \mathsf{T}_\mathsf{p}, \mathsf{C}_\mathsf{p}\rangle \in \Phi_s$, the support of ϕ_s is defined as the number of entities e satisfying ϕ_s, denoted as $e \models \phi_s$, hence:

$$\text{supp}(\phi_s) = |\{e \in \mathcal{I} \mid e \models \phi_s\}| \tag{B.1}$$

Finally, the confidence of a constraint ϕ_s measures the ratio between how many entities conform to ϕ_s and the total number of entities that are instances of the target class of the shape s.

Definition B.5 (Confidence of ϕ_s)
Given a shape $\langle s, \tau_s, \Phi_s\rangle \in \mathcal{S}$ having shape constraint $\phi_s{:}\langle \tau_\mathsf{p}, \mathsf{T}_\mathsf{p}, \mathsf{C}_\mathsf{p}\rangle \in \Phi_s$, the confidence of ϕ_s is defined as the proportion of entities for which $e \models \phi_s$ among the entities that are instances of the target class τ_s of $s \in \mathcal{S}$, hence:

$$\text{conf}(\phi_s) = \frac{\text{supp}(\phi_s)}{|\{e|(e, type, \tau_s) \in \mathcal{G}\}|} \tag{B.2}$$

As it happens in the case of frequent pattern mining [19], when extracting validating shapes, the support provides insights on how frequently a constraint is matched in the graph, i.e., the number of entities e satisfying a constraint ϕ_s. While similar to the task of itemset mining [6], the confidence can tell us how strong is the association between a node type and a specific constraint, i.e., the proportion of entities e satisfying a constraint ϕ_s among all the entities that are instances of the node type τ_s of $s \in \mathcal{S}$. For instance, the confidence for property shape headOf (Figure B.1b) in our snapshot of LUBM is 10% for the Full Professor node shape and 100% for Chair, which indicates a strong association of the headOf property shape to latter and a weak association to the former.

2.3 The Quality Shapes Extraction Problem

Given the need to extract shapes from a large existing graph $\mathcal{G}$ while limiting the effect of spuriousness, we formally define the problem of extracting high-quality shapes from KGs as follows:

Problem 1 (Quality Shapes Extraction)
Given an RDF graph $\mathcal{G}$, a threshold ω for support, and ε for confidence, the problem of quality shapes extraction over $\mathcal{G}$ is to find the set of shapes $\mathcal{S}$

such that for all node shapes $\langle s, \tau_s, \Phi_s \rangle \in \mathcal{S}$ it holds that $\text{supp}(s) > \omega$ and for all property shapes $\phi_s : \langle \tau_p, \mathsf{T}_p, \mathsf{C}_p \rangle \in \Phi_s$, $\text{supp}(\phi_s) > \omega$ and $\text{conf}(\phi_s) > \varepsilon$.

In the following, we provide both, an exact and an approximate solution to the problem of quality shape extraction.

3 QSE-Exact

Extracting shapes $\mathcal{S}$ from an RDF graph $\mathcal{G}$ requires processing its triples and analyzing the types of nodes involved both as subjects and objects in those triples. At a high level, we need to know for each entity all its types, these will become node shapes, and then for each entity type, identify property shapes, which requires, in turn, knowing the types of the objects as well. Furthermore, we need to keep frequency counts to know how often a specific property connects nodes of two given types compared to how many entities exist of those types. In our solution, this is done in four steps: (1) entity extraction, (2) entity constraints extraction, (3) support and Confidence computation, and (4) shapes extraction. Here we first consider the case where the graph is stored as a complete dump on a single file. Later, we also consider the case for a graph stored within a triplestore [41] for which the KG is not available as a file.

QSE-Exact (file-based). One of the most common ways to store an RDF graph $\mathcal{G}$ on a file **F** is to represent it as a sequence of triples. Therefore, QSE reads **F** line by line and processes it as a stream of $\langle s, p, o \rangle$ triples. Algorithm 1 and Figure B.2 present the four main steps of QSE to extract shapes for graph $\mathcal{G}$ stored in **F**. In the entity extraction phase, the algorithm parses each $\langle s, p, o \rangle$ triple containing a type declaration (e.g., rdf:type or wdt:P31 – this can be configured) and for each entity, it stores the set of its entity types and the global count of their frequencies, i.e., the number of instances for each class (Lines 4-8) in maps Ψ_{ETD} (Entity-to-Data) and Ψ_{CEC} (Class-to-Entity-Count), respectively. For example, Figure B.2 (phase 1) presents two example entities :bob and :alice (from the example graph of Figure B.1a) having entity types :Student, :FullProfessor, and :Chair, respectively. Figure B.2 also presents the structure of the Entity-to-Data Ψ_{ETD} dictionary map to help understand the captured entities and their information. In the second phase, i.e., entity constraints extraction, the algorithm performs a second pass over **F** (Lines 9-19) to collect the constraints and the meta-data required to compute support and confidence of each candidate property shape. Specifically, it parses all triples except triples containing type declarations (which can be skipped now) to obtain for each predicate the subject and object types from the map Ψ_{ETD} that was populated in the previous step. The type of a literal object is inferred from the value, and for a non-literal object is obtained from Ψ_{ETD} (Lines 11-16). For example, Ψ_{ETD} records that the types of :alice are :FullProfessor and

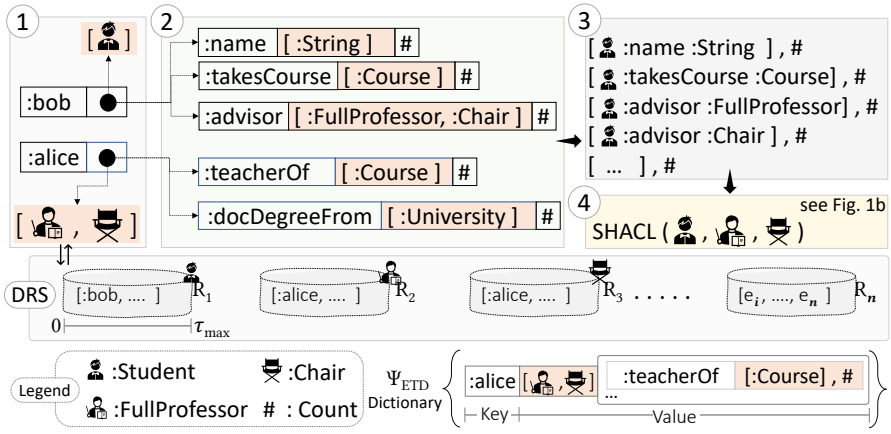


Fig. B.2: Overview of the four phases of QSE: ① entity extraction, ② entity constraints extraction, ③ support and confidence computation, and ④ shapes extraction. QSE-Approximate uses Dynamic Reservoir Sampling (DRS) in ①.

:Chair. Then, the Entity-to-Property-Data map Ψ_{ETPD} is updated to add the candidate property constraints associated with each subject entity (Line 17). Figure B.2 (phase 2) shows the meta-data captured for the properties of :bob and :alice.

In the third phase, i.e., for support and confidence computation, the constraints' information stored in maps (Ψ_{ETD}, Ψ_{CEC}) is used to compute support and confidence for specific constraints. The algorithm iterates over the map Ψ_{ETD} to get the inner map Ψ_{ETPD} mapping entities to candidate property shapes ϕ_s:$\langle \tau_p, \mathsf{T_p}, \mathsf{C_p} \rangle \in \Phi_s$, and retrieves the type of each entity using types information stored in Ψ_{ETD} to build triplets of the form $\langle \tau_e, \tau_p, \tau_{p_o} \rangle$ and compute their support and confidence (Line 25). Figure B.2 (phase 3) highlights some of these triplets for τ_e =:Student. The value of support and confidence for each distinct triplet is incremented in each iteration and stored in Ψ_{SUPP} and Ψ_{CONF} maps. Additionally, a map Ψ_{PTT} (Property to Types) is populated with distinct properties' frequencies and their object types in order to, later on, establish the corresponding min/max cardinality constraints (Line 26).

Finally, in the shapes extraction phase, the algorithm iterates over the values of the Ψ_{CTP} map and defines the *shape name* of s, the *shape's target definition* τ_s, and the set of *shape constraints* ϕ_s for each candidate class (Lines 27-29). The set of property shapes P for a given *Node Shape* are then extracted from the map $\text{MAP}\langle\text{PROPERTY}, \text{SET}\rangle$ (Lines 30-36). An example shapes graph for our running example is shown in Figure B.1. The $\mathsf{C_p}$ constraint can possibly have three types of values: sh:Literal, sh:IRI, and sh:BlankNode. In the case of literal types, the literal object types such as xsd:string, xsd:integer, or xsd:date are used. However, in the case of non-literal object types, the constraint

Algorithm 1 SHAPES EXTRACTION

Input: Graph $\mathcal{G}$ from File **F**, ω: min-support, ε: min-confidence
Output: Output: $\mathcal{S}\langle s, \tau_s, \Phi_s \rangle$

$E_{data} \leftarrow \{$T: SET$_{\text{Types}}$, Ψ_{ETPD} = Map$\langle$IRI, $P_{data}\rangle\}$
$P_{data} \leftarrow \{$T′ : SET$_{\text{ObjTypes}}$, Count :INT$\}$
Ψ_{ETD} = Map$\langle$IRI, $E_{data}\rangle$, Ψ_{CEC} = Map$\langle$IRI, INT$\rangle$, Ψ_{CTP} = Map$\langle$IRI, MAP$\langle$IRI, SET$\rangle\rangle$
for $t \in \mathcal{G} \wedge$ t.p = Type Predicate **do** ▷ ① **Entity extraction**
  entity e : t.s ; entityType e_t = t.o ▷ s: subject, o: object
  if $e \notin \Psi_{\text{ETD}}$ **then** Ψ_{ETD}.insert(e, ...))
  Ψ_{ETD}.insert(e, Ψ_{ETD}.get(e).T.add(e_t)) ▷ T : entity types
  increment entity count for current e_t in Ψ_{CEC}
for $t \in \mathcal{G} \wedge$ t.p != Type Predicate **do** ▷ ② **Entity constraints extraction**
  $\text{Set}_{\text{ObjTypes}} \leftarrow \emptyset$, $\text{Set}_{\text{TUPLE}} \leftarrow \emptyset$ ▷ init a *type* and *property to type* tuple set
  if object $t.o$ is Literal **then**
    $\text{Set}_{\text{ObjTypes}}$.add(getLiteralType(t.o))
    $\text{Set}_{\text{TUPLE}}$.add(new Tuple$\langle$ $t.p$, getLiteralType($t.o$)$\rangle$)
  else ▷ for non-literal objects
    for $\text{obj}_{\text{type}} \in \Psi_{\text{ETD}}$.get($t.o$).T **do**
      $\text{Set}_{\text{ObjTypes}}$.add($\text{obj}_{\text{type}}$) ; $\text{Set}_{\text{TUPLE}}$.add(new Tuple$\langle t.p, \text{obj}_{\text{type}}\rangle$)
  addPropertyConstraints($t.s$, $\text{Set}_{\text{TUPLE}}$, Ψ_{ETD})
  for IRI $\in \Psi_{\text{ETD}}$.get($t.s$.T) **do** ▷ if $t.s \in \Psi_{\text{ETD}}$
    update Ψ_{CTP} with class IRI, $t.p$, and object types using $\text{Set}_{\text{ObjTypes}}$
▷ ③ **Support and Confidence computation**
Ψ_{SUPP} = MAP$\langle$TUPLE$_3$, INT$\rangle$, Ψ_{CONF} = MAP$\langle$TUPLE$_3$, INT$\rangle$, Ψ_{PTT}
for $(e, E_{data}) \in \Psi_{\text{ETD}}$ **do**
  for $(T, \Psi_{\text{ETPD}}) \in E_{data}$ **do**
    for $e_t \in T \wedge (p, p_o, c) \in P_{data}$ **do**
      $\chi \leftarrow$ createTriplets($\langle \tau_e, \tau_p, \tau_{p_o} \rangle$)
      computeSupportAndConfidence (Ψ_{SUPP}, χ, Ψ_{CEC})
      computeMaxCardinality (Ψ_{PTT}, p , c)
for (class, MAP$\langle$Property, $\text{SET}_{\text{ObjTypes}}\rangle$) $\in \Psi_{\text{CTP}}$ **do** ▷ ④ **Shapes extraction**
  $\Phi_s \leftarrow \emptyset$ ▷ Property shapes $\Phi_s = \{\phi_{s_1}, \phi_{s_2}, ..., \phi_{s_n}\}$ where ϕ_s:$\langle \tau_p, \text{T}_p, \text{C}_p \rangle$
  s = class.buildShapeName() , τ_s = class
  for $(p, \text{SET}_{\text{ObjTypes}}) \in$ MAP$\langle$Property, SET$\rangle$ **do** $\phi_s.\tau_p = p$
    $p.\omega = \Psi_{\text{Supp}}$.get$(p, \text{SET}_{\text{ObjTypes}})$, $p.\varepsilon = \Psi_{\text{Conf}}$.get$(p, \text{SET}_{\text{ObjTypes}})$
    if $p.\omega > \omega \wedge p.\varepsilon > \varepsilon$ **then**
      build(sh:nodeKind, sh:maxCount, Ψ_{PTT})
    $\phi_s.\text{C}_p.add$(sh:minCount : 1) ▷ $if\ p.\varepsilon > \varepsilon'$
    $\Phi_s.add(\phi_s)$
  $\mathcal{S}.add(s, \tau_s, \Phi_s)$ ▷ if s.$\omega > \omega \wedge \phi_s !\emptyset$

sh:class is used to declare the type of object to define the type of value for the candidate property. It is possible to have more than one value for the sh:class and sh:datatype constraints of a candidate property shape, e.g., to state that a property can accept both integers and floats as values, in such cases, we use sh:or constraint to encapsulate multiple values. A detailed explanation of each phase is available in the extended version of the paper[1].

[1] https://relweb.cs.aau.dk/qse/

QSE-Exact (query-based). To support shapes extraction from a triple-store, we propose QSE-Exact query-based that uses a set of SPARQL queries [40] to extract all the necessary information that we collect across the four phases. In practice, we pose queries to extract all the distinct classes $\mathcal{C}$, then, for each class $c \in \mathcal{C}$, its properties $p \in \mathcal{P}$ along with object types are extracted as triplets, and support is computed for each triplet by a count query. This method is based on the standard procedure also implemented in other existing, query-based tools [12, 20].

Cardinality Constraints. QSE supports assigning cardinality constraints (sh:minCount and sh:maxCount) to $\mathsf{C_p}$ to each property shape constraint $\phi_s{:}\langle \tau_\mathsf{p}, \mathsf{T_p}, \mathsf{C_p} \rangle$. Following the open-world assumption, all shape constraints are initially assigned a minimum cardinality of 0, making them optional. However, there are cases where we can infer that some properties are mandatory (i.e., should be assigned a min count of 1), and some other properties should appear exactly once for each entity (i.e., should be assigned both a min and a max count equal to 1). Trivially one can assign minimum cardinality 1 to property shapes having confidence 100%, i.e., for those cases in which all entities have that property. In case of incomplete KGs, QSE allows users to provide a different confidence threshold value for adding the min cardinality constraints. To achieve this, we extend the fourth phase and add a min cardinality constraint in property shapes on line 35 based on the min-confidence provided by the user. QSE also keeps track of properties having maximum cardinality equal to 1 in a second phase and assigns sh:maxCount=1 to those property shapes in the fourth phase of shapes extraction.

Complexity Analysis. The time complexity of QSE-Exact (Algorithm 1) is $\mathcal{O}(2{\cdot}|F| + |E|{\cdot}|\Phi_s| + |S|{\cdot}|\Phi_s|)$. Where $2 * |F|$ refers to the first and second phases having to parse all the triples twice, E is the set of entities (i.e., the set of distinct IRIs that appear as a subject for some triple), S is the set of Node Shapes, and lastly, Φ_s represents a set of all property shape constraints, i.e., $\Phi_s = \{\phi_1, \phi_2, ..., \phi_n\}$. Therefore, our algorithm scales linearly in the number of edges and nodes in the graph and in the size of the final set of shapes.

4 QSE-Approximate

QSE-Exact keeps type and property information for each entity in memory while extracting shapes. As a result, its memory requirements are prohibitively large when dealing with large KGs. Therefore, we propose QSE-Approximate to enable shape extraction from very large KGs with reduced memory requirements. Our goal is to *solve the scalability issue in shapes extraction approaches by using only the resources available to a commodity machine.* QSE-Approximate is based on a multi-tiered dynamic reservoir-sampling (DRS) algorithm that we introduce. We maintain as many reservoirs as types in the

graph, and we dynamically resize each reservoir as new triples are parsed. Moreover, the replacement of nodes in the reservoir is performed based on the number of node types across reservoirs. The resulting algorithm replaces the first phase of QSE. After sampling, the information about the sampled entities is used in the same way as before in the remaining phases of Algorithm 1. Hence, we maintain information only for a small representative sample of entities in memory but enough to detect all shapes.

Algorithm 2 receives as input a graph file **F**, sampling percentage (Sampling$_{\%}$), and maximum size of the reservoir per class (τ_{max}). After initialization, triples t of **F** are parsed (Line 3) and filtered based on whether they contain a type declaration. From these, we extract the entities to populate the Entity-to-Data map Ψ_{ETD} (Lines 4-24), while non-type triples are parsed on Line 24 to keep count of distinct properties in the Property-Count map Ψ_{PC}. For instance, :alice is an entity of type :FullProfessor and :Chair in Ψ_{ETD} shown in Figure B.2. QSE-Approximate maintains a reservoir for each distinct entity type e_t, e.g., maintaining a distinct reservoir of entities of type :Student (R_1), :FullProfessor (R_2), and :Chair (R_3) shown in Figure B.2, using a map of sampled entities per class (Ψ_{SEPC}). The reservoir capacity map (Ψ_{RCPC}) stores the current max capacities for the reservoir for each e_t. If e_t does not exist in Ψ_{SEPC} and Ψ_{RCPC}, i.e., if it has not a reservoir, one is created (lines 6-7). Then, e is inserted in the reservoir for e_t (Lines 8-11), e.g., :alice is inserted into both reservoirs R_2 and R_3 shown in Figure B.2. If the reservoir has reached its current capacity limit, we may have to replace an entity in the reservoir with the current one. Hence, neighbor-based dynamic reservoir sampling is performed (Lines 13-18), i.e., a random number r is generated between zero and the current number of type declarations read from **F**. If r falls within the reservoir size, then a node in the reservoir is replaced with e. To select which node to replace, we identify as $\widehat{n}$ the target node at index r, and with $\overleftarrow{n}$ and $\vec{n}$ its neighbors at indexes $r-1$ and $r+1$, respectively. Among these, the node having minimum scope (i.e., the minimum number of types that are known at this point in time) is selected to be replaced by the current e (Line 17). Additionally, the algorithm keeps track of actual Class-to-Entity-Count in Ψ_{CEC} (Line 19), i.e., the exact count of how many entities of each type we have seen. Once the reservoir for e_t is updated, the sampling ratio for this type is computed, i.e., the proportion of entities kept so far with type e_t over the total number of entities of that type seen up to now. Given the current and target sampling ratio (Sampling$_{\%}$) provided as input, the algorithm evaluates whether to resize the reservoir for e_t, if it has not already reached the limit τ_{max} (Lines 21-23).

While performing shapes pruning using counts over sampled entities, QSE-Approximate requires to estimate actual support $\overline{\omega}_\phi$ and confidence $\overline{\varepsilon}_\phi$ of a property shape ϕ from the current values ω and ε computed from the sampled data. Thus, it estimates with $\overline{\omega}_\phi = \omega_\phi / min(|P_r^*|/|P|, |T_r|/|T|)$ the

Algorithm 2 QSE-Approximate Reservoir Sampling

Input: Graph $\mathcal{G}$ from File **F**, maximum entity threshold τ_{max}, Sampling$_{\%}$
Output: Ψ_{ETD} , Ψ_{CEC}

init maps Ψ_{ETD} , Ψ_{SEPC}, Ψ_{RCPC} , Ψ_{CEC}, Ψ_{PC}
$\tau_{\mathsf{min}} = 1$ (minimum entity threshold) ; lineCounter $= 0$
for t $\in \mathcal{G}$ **do** ▷ parse s,p,o of the triple t
  if t.p $=$ Type Predicate **then**
    entity e : t.s ; entityType e_t = t.o ▷ s: subject, o: object
    Ψ_{SEPC}.putIfAbsent(e_t, [ ]) ▷ if $e_t \notin \Psi_{\text{SEPC}}$
    Ψ_{RCPC}.putIfAbsent(e_t, τ_{min}) ▷ if $e_t \notin \Psi_{\text{RCPC}}$
    if $\mid \Psi_{\text{SEPC}}$.get (e_t) $\mid < \Psi_{\text{RCPC}}$.get ($e_t$) **then** ▷ Add entity e in reservoir
      if Ψ_{ETD}.get(e).T is $\emptyset$ **then** Ψ_{ETD}.insert(e, ...)) ▷ T : entity types
      Ψ_{ETD}.insert(e, Ψ_{ETD}.get(e).T.add(t.o))
      Ψ_{SEPC}.get(e_t).insert(e)
    else ▷ Replace random entity in reservoir with current entity e
      r = generateRandomNumber(0, lineCounter)
      if $r < \mid \Psi_{\text{SEPC}}$.get($e_t$) $\mid$ **then**
        $\bar{n}, \hat{n}, \vec{n} = \Psi_{\text{SEPC}}$.get($e_t$).nodeAtIndex($r-1, r, r+1$))
        n = getNodeWithMinimumScope($\bar{n}, \hat{n}, \vec{n}$)
        replace node at index n with current e & e_t in Ψ_{ETD}
        Ψ_{SEPC}.get(e_t).add (e)
    increment entity count for current e_t in Ψ_{CEC}
    ▷ Resize reservoir
    ratio = ($\Psi_{\text{SEPC}}.get(e_t).size()/\Psi_{\text{CEC}}.get(e_t)$) $\times$ 100
    capacity = Sampling$_{\%}$ $\times$ Ψ_{SEPC}.get(e_t).size()
    if capacity $< \tau_{max} \wedge$ ratio $\leq$ Sampling$_{\%}$ **then** Ψ_{RCPC}.insert(e_t , capacity)
  else$\rightarrow$ increment property count for current $t.p$ in Ψ_{PC}
  lineCounter $++$

effective support for a property shape ϕ, where ω_ϕ is the support computed for ϕ in the current sample, P represents all triples in $\mathcal{G}$ having property τ_p, P_r^* represents triples having property τ_p across all entities in all reservoirs, T represents all entities of type e_t in $\mathcal{G}$, and T_r represents all entities of type e_t in the reservoir. Similarly, the confidence $\bar{\varepsilon}_\phi$ of a property shape is estimated by replacing denominator in eq. (B.2) with $|T_r|$.

QSE-Approximate (query-based). We apply the same sampling technique in the query-based shapes extraction approach where in Algorithm 2 entities and their meta-data are retrieved via SPARQL queries, resulting in *query-based* QSE-Approximate.

Space Analysis. The space requirement of QSE-Approximate depends on the values of target Sampling$_{\%}$, the maximum reservoir size τ_{max}, and the number of entity types $\mid$T$\mid$ in $\mathcal{G}$. In the worst case, it requires $O(2{\cdot}|T|{\cdot}\tau_{max})$, therefore while $\mathcal{G}$ can contain hundreds of millions of entities, we can still easily estimate how many distinct types are in the graph and select τ_{max} to fit the available memory.

5 Evaluation

In the following, we evaluate our QSE solutions and their effectiveness in tackling the problem of *spuriousness* along with a comparison to existing state-of-the-art approaches.

Datasets. We selected a synthetic dataset, LUBM-500 [18], and three real-world datasets: DBpedia [4] downloaded on 01.10.2020; YAGO-4 [46], for which we use the subset containing instances from the English Wikipedia, downloaded on 01.12.2020; and WikiData [49], in two variants, i.e., a dump from 2015 [54] (Wdt15), used in the original evaluation of SheXer [12], and the truthy dump from September 2021 (Wdt21) filtered by removing non-English strings. Table B.1 provides a comparison of their contents.

Experimental Setup. We have implemented QSE algorithms in JAVA-11. All experiments are performed on a single machine with Ubuntu 18.04, having 16 cores and 256 GB RAM. We have used GraphDB [16] 9.9.0 to experiment with *query-based* variants of QSE with a maximum memory usage limit of 16 GB. The source code of QSE is available as open-source [37] along with experimental settings and datasets. We have also published the extracted SHACL shapes of all our datasets on Zenodo [39]. For SheXer, we cloned its original code from GitHub and used the same settings as the original paper, i.e., default tuned parameters for the sheXing process and customized tuned parameters to output shapes equivalent to QSE.

Metrics. We measure the *running time*, and maximum *memory usage* (defined using Java `-Xmx`) during QSE shapes extraction process, and *Shape Statistics* of the output shapes.

QSE-Exact. We use QSE-Exact to extract shapes from LUBM (L), DBpedia (D), YAGO-4 (Y), and WikiData (W). The statistics of the shapes extracted

Table B.1: Size and characteristics of the datasets

	DBpedia	**LUBM**	**YAGO-4**	**Wdt15**	**Wdt21**
# of triples	52 M	91 M	210 M	290 M	1.926 B
# of objects	19 M	12 M	126 M	64 M	617 M
# of subjects	15 M	10 M	5 M	40 M	196 M
# of literals	15 M	5.5 M	111 M	40 M	904 M
# of instances	5 M	1 M	17 M	3 M	91 M
# of classes	427	22	8,902	13,227	82,693
# of properties	1,323	20	153	4,906	9,017
Size in GBs	6.6	15.66	28.59	42	234

from these datasets using QSE-Exact (file-based) are shown in Table B.2. It shows the count of Node Shapes (NS), Property Shapes (PS), and Property Shape Constraints (PSc), i.e., literal and non-literal node types constraints. We refer to these statistics as *default shape statistics.* We initially considered SheXer [12], ShapeDesigner [5], and SHACLGEN [20] as state-of-the-art approaches [38] to compare against QSE. Among these, both ShapeDesigner and SHACLGEN load the whole graph into a triplestore similar to our QSE-Exact (query-based). Yet, their current implementations cannot handle large KGs with more than a few million triples and do not manage to extract shapes of KGs having more than some hundreds of classes. In our experiments, either they crashed because they tried to load the graph into an in-memory triplestore or required multiple hours to generate shapes for large KGs such as YAGO-4 (with 8,897 classes). Therefore, in the following, we focus our comparison on SheXer, which supports both the file-based and the query-based methods. Table B.3 shows the running time and memory consumption to extract shapes for all datasets using File (**F**) and Query-based (**Q**) variants of SheXer, QSE-Exact, and QSE-Approximate. Among the *file-based* approaches, QSE-Exact is 1 order of magnitude faster than SheXer for all datasets. It consumes up to 50% less memory than SheXer to extract shapes from D, L, Y, and Wdt15, whereas SheXer goes out of memory ($\mathsf{Out_M}$) for Wdt21. Similarly, among the *query-based* approaches, QSE-Exact is 1 order of magnitude faster and consumes less than 50% memory to extract shapes from D, Y, L, and Wdt15. SheXer timed out ($\mathsf{Out_T}$ – 24 hours) for Y and Wdt21, while QSE-Exact timed out for Wdt21 only.

Taming spuriousness. To deal with the issue of spuriousness, we analyze the shapes extracted and kept after pruning. QSE performs *support-based shapes extraction* by producing only the shapes with support and confidence greater than or equal to a threshold specified by the user. For instance, given a minimum support threshold of 100 and minimum confidence value 25%, for every PS, QSE prunes all the PSc that do not appear with at least 100 entities or if not at least for 25% of entities for that type. We remind that

Table B.2: Shapes Statistics using QSE-Exact.

	NS	**PS**	**Non-Literal PSc**	**Literal PSc**
	COUNT	**COUNT/AVG**	**COUNT/AVG**	**COUNT/AVG**
LUBM	23	164 / 7.1	323 / 3.0	57 /1.0
DBpedia	426	11,916 / 27.9	38,454 / 6.9	5,335 /1.0
YAGO-4	8,897	76,765 / 8.6	315,413 / 14.5	50,708 / 1.0
Wdt15	13,227	202,085 / 15.2	114,890 / 3.0	106,599 / 1.0
Wdt21	82,651	2,051,538 / 24.8	3,765,953 / 5.6	1,113,856 / 1.0

Table B.3: Running Time (T) in minutes (m) and hours (h) along with Memory (M) consumption in GB.

		DBpedia		LUBM		YAGO-4		Wdt15		Wdt21	
		T	M	T	M	T	M	T	M	T	M
F	SheXer	26 m	18	58 m	33	1.9 h	24	3.2 h	59	-	Out_M
	QSE-Exact	3 m	16	8 m	16	23 m	16	16 m	50	2.5 h	235
	QSE-Approx	**1 m**	10	**2 m**	10	**13 m**	10	**13 m**	16	**1.3 h**	32
Q	SheXer	9 h	65	15 h	140	Out_T	-	13 h	180	Out_T	-
	QSE-Exact	34 m	16	47 m	16	2.4 h	16	1.2 h	16	Out_T	-
	QSE-Approx	**16 m**	6	**3 m**	7	**39 m**	16	**49 m**	16	**5.7 h**	64

the pruning of PSc has a cascading effect that also affects the pruning of PS, and the pruning of PS can, in turn, cause the pruning of NS. To study the impact of various confidence and support thresholds on the number of PSc, PS, and NS, we analyze the effect of pruning by specifying various values for confidence and support. Figure B.3 shows the result of pruning PSc (B.3a,b), PS and NS (B.3c,d) for confidence >(25, 50, 75,90)% and support ($\geq$1, >100) on DBpedia and Wdt21. Experimental results on LUBM, YAGO-4, and Wdt15 are comparable to the results presented for DBpedia and Wdt21, and are reported in the extended version of the paper[1]. In general, as expected, the results show that the higher we set the threshold for support and confidence, the higher the percentage of PSc and PS to be pruned. Precisely, DBpedia contains 11K PS, 38K non-literal, and 5K literal PSc (Table B.2), when QSE performs pruning with confidence >25% and support $\geq$ 1, it prunes out 99% PSc and PS (Figure B.3a,b). Similarly for Wdt21, QSE prunes 85% non-literal and 97% literal constraints, and 66% PS for confidence >25% and support $\geq$1 (Figure B.3b). In comparison to the default shape statistics (Table B.2), increasing confidence to >50%, 75%, and 90%, pruning resulted in a drastic

Table B.4: QSE-Approximate: Effect of $Sampling_{\%}$ (S%) and reservoir size (τ_{max}) on Precision (P), Recall (R), and Relative Error (Δ) with min. support 1 and confidence 25% on Wdt21

S%	τ_{max}	**Property Shapes (PS)**				**Time (Min)**	**Mem (GB)**
		Real	**Sample**	**P / R**	Δ		
10%	20	698,825	470,562	1.00 / 0.61	228,263	81	16
	200	698,825	497,035	0.92 / 0.65	201,790	81	16
50%	500	698,825	548,381	0.96 / 0.79	150,444	82	24
	5000	698,825	605,785	0.96 / 0.83	93,040	95	24
100%	500	698,825	617,349	1.00 / 0.88	81,476	87	32
	5000	698,825	645,810	1.00 / 0.92	53,015	98	32

decrease in the number of PSc and PS. In DBpedia, the majority of non-literal PSc are pruned out, and in Wdt21, the majority of literal constraints are pruned out. Pruning of NS is lower compared to PS and PSc for all combinations of support and confidence, showing that almost all types are associated at least with some very common PSc, e.g., the fact to have a :name.

QSE-Approximate. The QSE-Approximate approach reduces the memory requirements of the exact approach by allowing users to specify the *sampling percentage* (Sampling$_{\%}$, S% for short) and maximum limit of the *reservoir size* (τ_{max}), i.e., the maximum number of entities to be sampled per class, to reduce the number of entities to keep in memory. Table B.3 shows that among the *file-based* approaches, QSE-Approximate is the most efficient approach compared to QSE-Exact and SheXer. For example, to extract shapes from Wdt21, QSE-Approximate (with $\tau_{max} = 1000$ and S%=100%) required almost half the time with 1 order of magnitude less memory than QSE-Exact, while SheXer could not complete the computation. Similarly, among *query-based* approaches, QSE-Approximate proved to be the only approach to extract shapes from the Wdt21 endpoint in 5.7 hours with 64 GB memory consumption. In contrast, QSE-Exact and SheXer timed out (24 hours). Analogously to Wdt21, QSE-Approximate remains 1 order of magnitude faster with 50% less memory consumption than SheXer (for both query and file-based variants) to extract shapes from D, L, Y, and Wdt15. Overall, these results show that our proposals have solved scalability issues in shape extraction approaches regardless of the type of input data source (file or endpoint). The choice of using a query-based or file-based version depends on the given setting. For instance, querying an endpoint to extract shapes can impose excessive stress on a production DBMS serving other applications. On the other hand, the file-based approach is less resource-intensive and can be used if the user can afford the cost of dumping the graph into a file.

QSE Sampling Parameters. We further evaluate the quality of the output of QSE-Approximate using multiple combinations of values for S% and τ_{max} on Wdt21 with a fixed confidence and support threshold. This analysis helps the user to choose the best values for S% and τ_{max} parameters given some memory constraints. We show the results in Table B.4, where the values shown in columns *Real* and *Sampled* are extracted by QSE-Exact and QSE-Approximate, respectively. Here we skip listing values for NS as they are not affected by the values of S%, τ_{max}, confidence, and support. The results show that S%=10 and τ_{max} up to 200 provide a 92% precision for PS extracted using QSE-Approximate and pruned with support $\geq$1 and confidence >25%. This requires only 16 GB RAM and 81 minutes. If a machine up to 24 GB RAM is available, then S%=50% and τ_{max}=5K provide 96% precision with $\Delta = 93$K in 95 minutes. Similarly, on a machine having 32 GB RAM, S%=100% and τ_{max}=5K provide 100% precision with $\Delta = 53$K in 98 minutes. The non-perfect precision translates into some shapes being produced despite their

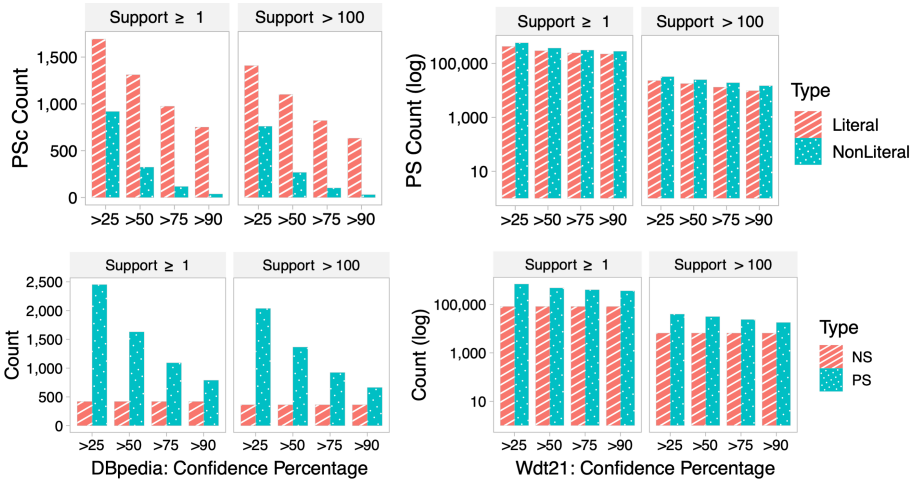


Fig. B.3: QSE-Exact on DBpedia and Wdt21

support and confidence is slightly lower than required. We also see that, for very small values of τ_{max} we achieve a lower recall, meaning that some shapes that should have been produced are instead wrongly pruned. We note though that min support 1 and confidence 25% are still quite low values and the shapes produced are thus more affected by spuriousness. Nonetheless, on a standard commodity machine with 32GB we see we can easily achieve perfect precision (100%) and very high recall (92%).

We further study the effect of pruning on shapes extracted from Wdt21 using QSE-Approximate with confidence >25% and >75% having support 1, 10, and 100 (shown in Table B.5). We see that with support $\geq$1 and confidence $>$25%, QSE-approximate is able to get almost all the PS extracted by QSE-Exact for Wdt21 (Figure B.3d) having 89% recall and 100% precision. Additionally, upon increasing the support to 10 and 100, we notice a constant recall of around 88-99% and a slight reduction in precision, i.e., 98% and 96% with decreasing relative error (i.e., Δ). Similarly, we notice the same trend with confidence=75%. Therefore, while we very rarely overestimate the support and confidence of the shapes produced, we underestimate some of these values, although still in a few cases only.

Practical Implications of QSE. We show the practical utility of QSE by evaluating the correctness of extracted shapes and their effect when used to validate the KG. We extracted shapes from DBpedia using QSE with confidence >25% and support >100. Then, we randomly selected 10 shapes and manually inspected them to evaluate their correctness, i.e., whether these shapes describe valid constraints. This allows us to measure precision and recall based on the pruning parameters. The results of this analysis showed

Table B.5: Output quality of QSE-Approximate on Wdt21 with S% = 100% and $\tau_{max} = 500$ as # of real and sampled NS, PS, and corresponding Precision (P), Recall (R), and Relative Error Δ.

Conf	Supp	Node Shapes (NS)				Property Shapes (PS)			
		Real	Sample	P / R	Δ	Real	Sample	P / R	Δ
> 25 %	≥ 1	82,651	82,651	1.0 / 1.0	0	698,825	620,622	1.00 / 0.89	78,203
	10	23,640	23,640	1.0 / 1.0	0	158,283	141,040	0.99 / 0.88	17,243
	100	6,596	6,596	1.0 / 1.0	0	39,877	36,362	0.96 / 0.88	3,515
> 75 %	≥ 1	82,651	82,651	1.0 / 1.0	0	405,344	362,717	1.00 / 0.89	42,627
	10	23,640	23,640	1.0 / 1.0	0	91,947	83,329	0.99 / 0.90	8,618
	100	6,596	6,596	1.0 / 1.0	0	23,944	22,193	0.97 / 0.90	1,751

that QSE extracts shapes with 100% precision in terms of correct shapes constraints that should be part of the final set of shapes (qualified as quality shapes) by removing spurious shape constraints. Further, we used these 10 shapes, extracted by QSE, to validate DBpedia using a SHACL validator and found 20,916 missing triples and 155 erroneous triples. The detailed results of this analysis are contained in the extended version[1]. Overall, this experiment shows that by using our technique the user is provided with a refined set of valid shapes that can effectively identify errors in the KG.

Constraints Coverage. Comparing the constraints supported by QSE and existing approaches (i.e., SheXer [12], SHACLGEN [20], and ShapeDesigner [5]), we report that QSE is able to extract the widest range of constraints (i.e., 15 out of 16 specific core constraints). Amongst those that are usually not supported, we support sh:in, sh:Literal, sh:class, sh:not, and sh:node. We currently do not support sh:inverse but we plan to support it in the future. More details are available in the extended version of our paper[1].

Optimal Pruning Thresholds. For each class in the KG, QSE computes its frequency. Thus, this information can be used as a reference for the support and confidence thresholds. Further, QSE also supports the extraction of shapes for specific classes only. Therefore, the user can make use of frequency information and set class-specific pruning thresholds.

6 Related Work

KG Data Validation. Integrity constraints for KGs were initially defined with the RDF schema vocabulary [11] and then with the OWL language [27, 28, 47]. Later, the SPARQL Inferencing Notation (SPIN) [22] was proposed. SHACL [23] (a W3C standard since 2017) is known as the next generation of SPIN. Similar to SHACL, ShEX [35] is a constraint language that is built on regular bag expressions inspired by schema languages for XML.

Table B.6: State-of-the-art to extract validating shapes [38]

Approach	*Extracted from*		*Auto-matic*	*Triple-store*	*Type*
	data	*ontology*			
Shape Induction [26]	✔	✗	✔	✔	SHACL,ShEx
SheXer [12]	✔	✗	✔	✔	SHACL,ShEx
Spahiu et al. [45]	✔	✗	✔	✔	SHACL
ShapeDesigner. [5]	✔	✗	✔	✔	SHACL,ShEx
SHACLGEN [20]	✔	✔	✔	✔	SHACL
TopBraid [36]	✔	✔	✔	✔	SHACL
Pandit et al. [32]	✗	✔	✗	✔	SHACL
Astrea [9]	✗	✔	✔	✗	SHACL
SHACLearner [30]	✔	✗	✔	✗	SHACL
Groz et al. [17]	✔	✗	✔	✗	ShEx

While ShEx is not a standard, it is used within the WikiData project [48]. Even though SHACL and ShEx are not completely equivalent [15], their core mechanism revolves around the same concept of enforcing for each node to satisfy specific constraints on the combination of its types and predicates [12]. In this work, we support the extraction of validating shapes that can be represented in both languages.

Shape Extraction. Given the abundance of large-scale KGs, various applications have been created to assist the process of extracting information about its implicit or explicit schema [21]. Among these, shapes construction or extraction approaches, i.e., to generate a set of shapes given information from an existing KG, are used in order to obtain validating schema to ensure the quality of a KG's content. We have classified existing approaches in Table B.6 based on their features, i.e., support for shapes extraction from data or ontologies, support for automatic extraction of shapes, support for shapes extraction from a SPARQL triplestore, and whether they extract SHACL, ShEx, or both types of validating shapes. In our recent community survey [38] on extraction and adoption of validating shapes, we show that *there is a growing need among practitioners for techniques for efficient extraction of validating shapes from very large existing KGs*. Note that there exist approaches for schema extraction from property graphs as well [24]. Such approaches are not directly applicable to RDF KGs since their schema is more complex, moreover they focus on identifying sub-types based on node labels (which do not exist in RDF data, since types are nodes in the graph), and finally are not designed to handle the issue of spuriousness. Once shapes are extracted, they can be used to validate KGs using validation approaches like MagicShapes [2] and Trav-SHACL [13].

Rules, Patterns, and Summaries. There exist various approaches for rule discovery in graphs [25]. These systems [1, 14, 31] derive rules from

large KGs using structural information by exploring the frequently occurring graph patterns. In contrast to validating shapes, rules are mainly used to derive new facts from an incomplete KG or identify specific sets of wrong connections. Frequent subgraph mining (FPM) approaches, instead, are designed to find frequently recurring structures in a large graph. In FPM, the occurrence of subgraphs (the number of times a subgraph appears) cannot be taken as the support of subgraphs since it does not satisfy the non-monotonic property [7]. The most practical measurement for measuring this support is, instead, the minimum image-based support (MNI [7]). Our proposed definition of support for shape constraints is inspired by the concept of MNI support and its use in FPM [19]. Yet, different than FPM, we do not extract patterns of arbitrary shape and size, thus we are able to provide better performance guarantees as we solve a simpler problem. Finally, our approach is also related to the techniques of graph summarization [8] and can be seen as a special form of structural summarization [33]. Additionally, QSE provides a scalable solution for understanding the content of large KGs (by extracting their shapes) like ABSTAT-HD [3], which is based on exploring semantic profiles of large KGs.

7 Conclusion

In this paper, we propose an automatic shape extraction approach that addresses the two common limitations in other existing techniques, i.e., scalability and spuriousness. We addressed these limitations by introducing the Quality Shapes Extraction (QSE) problem. We devised an exact and approximate solution for QSE to enable the efficient extraction of shapes on commodity machines. Our method is based on the well-understood concepts of support and confidence, hence it allows a data scientist to focus on the shapes providing the highest reliability first when addressing issues of data quality. By setting even low pruning thresholds, QSE can prune up to 93% of the shapes that a trivial extraction would produce (i.e., a reduction of 2 orders of magnitude), shapes that hence have little support from the data and are thus likely spurious. Furthermore, we show that our approximate technique introduces only negligible loss in the quality and completeness of the produced shapes. In the future, we will extend the scope of constraints covered by QSE and a solution to automatically learn the optimal configurations for pruning thresholds for QSE.

References

[1] N. Ahmadi, T.-T.-D. Truong, L.-H.-M. Dao, S. Ortona, and P. Papotti, "Rulehub: A public corpus of rules for knowledge graphs," *Journal of Data and Information Quality (JDIQ)*, vol. 12, no. 4, pp. 1–22, 2020.

[2] S. Ahmetaj, B. Löhnert, M. Ortiz, and M. Šimkus, "Magic shapes for shacl validation," *Proceedings of the VLDB Endowment*, vol. 15, no. 10, pp. 2284–2296, 2022.

[3] R. A. Alva Principe, A. Maurino, M. Palmonari, M. Ciavotta, and B. Spahiu, "Abstat-hd: a scalable tool for profiling very large knowledge graphs," *The VLDB Journal*, vol. 31, no. 5, pp. 851–876, 2022.

[4] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives, "Dbpedia: A nucleus for a web of open data," in *The Semantic Web, 6th International Semantic Web Conference*, ser. Lecture Notes in Computer Science, vol. 4825. Busan, Korea: Springer, 2007, pp. 722–735.

[5] I. Boneva, J. Dusart, D. Fernández-Álvarez, and J. E. L. Gayo, "Shape designer for shex and SHACL constraints," in *Proceedings of the ISWC 2019 Satellite Tracks*, ser. CEUR Workshop Proceedings, vol. 2456. Auckland, New Zealand: CEUR-WS.org, 2019, pp. 269–272.

[6] C. Borgelt, "Frequent item set mining," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, vol. 2, no. 6, pp. 437–456, 2012.

[7] B. Bringmann and S. Nijssen, "What is frequent in a single graph?" in *PAKDD*, ser. Lecture Notes in Computer Science, vol. 5012. Osaka, Japan: Springer, 2008, pp. 858–863.

[8] Š. Čebirić, F. Goasdoué, H. Kondylakis, D. Kotzinos, I. Manolescu, G. Troullinou, and M. Zneika, "Summarizing semantic graphs: a survey," *The VLDB journal*, vol. 28, no. 3, pp. 295–327, 2019.

[9] A. Cimmino, A. Fernández-Izquierdo, and R. García-Castro, "Astrea: Automatic generation of SHACL shapes from ontologies," in *ESWC*, ser. Lecture Notes in Computer Science, vol. 12123. Heraklion, Crete, Greece: Springer, 2020, p. 497.

[10] W. Consortium, "RDF 1.1," https://w3.org/RDF/, 2014, accessed 17th January, 2023.

[11] ——, "RDF Schema 1.1," https://www.w3.org/TR/rdf-schema/, 2014, accessed 17th January, 2023.

[12] D. Fernandez-Álvarez, J. E. Labra-Gayo, and D. Gayo-Avello, "Automatic extraction of shapes using shexer," *Knowledge-Based Systems*, vol. 238, p. 107975, 2022.

[13] M. Figuera, P. D. Rohde, and M. Vidal, "Trav-shacl: Efficiently validating networks of SHACL constraints," in *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*. Ljubljana, Slovenia: ACM / IW3C2, 2021, pp. 3337–3348. [Online]. Available: https://doi.org/10.1145/3442381.3449877

[14] L. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek, "Fast rule mining in ontological knowledge bases with amie+," *The VLDB Journal*, vol. 24, no. 6, pp. 707–730, 2015.

[15] J. E. L. Gayo, E. Prud'Hommeaux, I. Boneva, and D. Kontokostas, "Validating rdf data," *Synthesis Lectures on Semantic Web: Theory and Technology*, vol. 7, no. 1, pp. 1–328, 2017.

[16] GraphDB, "GraphDB," https://graphdb.ontotext.com, 2023, accessed 17th January, 2023.

[17] B. Groz, A. Lemay, S. Staworko, and P. Wieczorek, "Inference of shape graphs for graph databases," in *25th International Conference on Database Theory, ICDT 2022, March 29 to April 1, 2022, Edinburgh, UK (Virtual Conference)*, ser. LIPIcs, vol. 220. Edinburgh, UK: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 14:1–14:20. [Online]. Available: https://doi.org/10.4230/LIPIcs.ICDT.2022.14

[18] Y. Guo, Z. Pan, and J. Heflin, "Lubm: A benchmark for owl knowledge base systems," *Journal of Web Semantics*, vol. 3, pp. 158–182, 2005.

[19] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: current status and future directions," *Data mining and knowledge discovery*, vol. 15, no. 1, pp. 55–86, 2007.

[20] A. Keely, "SHACLGEN," https://pypi.org/project/shaclgen/, 2023, accessed 17th January, 2023.

[21] K. Kellou-Menouer, N. Kardoulakis, G. Troullinou, Z. Kedad, D. Plexousakis, and H. Kondylakis, "A survey on semantic schema discovery," *VLDB J.*, vol. 31, no. 4, pp. 675–710, 2022. [Online]. Available: https://doi.org/10.1007/s00778-021-00717-x

[22] H. Knublauch, J. A. Hendler, and K. Idehen, "Spin-overview and motivation," *W3C Member Submission*, vol. 22, p. W3C, 2011.

[23] H. Knublauch and D. Kontokostas, "Shapes constraint language (shacl)," *W3C Candidate Recommendation*, vol. 11, no. 8, 2017.

[24] H. Lbath, A. Bonifati, and R. Harmer, "Schema inference for property graphs," in *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*. Nicosia, Cyprus: OpenProceedings.org, 2021, pp. 499–504. [Online]. Available: https://doi.org/10.5441/002/edbt.2021.58

[25] M. Loster, D. Mottin, P. Papotti, J. Ehmüller, B. Feldmann, and F. Naumann, "Few-shot knowledge validation using rules," in *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*. Ljubljana, Slovenia: ACM / IW3C2, 2021, pp. 3314–3324. [Online]. Available: https://doi.org/10.1145/3442381.3450040

[26] N. Mihindukulasooriya, M. R. A. Rashid, G. Rizzo, R. García-Castro, Ó. Corcho, and M. Torchiano, "RDF shape induction using knowledge base profiling," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC*. Pau, France: ACM, 2018, pp. 1952–1959.

[27] B. Motik, I. Horrocks, and U. Sattler, "Adding integrity constraints to OWL," in *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*, ser. CEUR Workshop Proceedings, vol. 258. Austria: CEUR-WS.org, 2007.

[28] ——, "Bridging the gap between owl and relational databases," *Journal of Web Semantics*, vol. 7, no. 2, pp. 74–89, 2009.

[29] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor, "Industry-scale knowledge graphs: lessons and challenges," *Communications of the ACM*, vol. 62, no. 8, pp. 36–43, 2019.

[30] P. G. Omran, K. Taylor, S. J. R. Méndez, and A. Haller, "Towards SHACL learning from knowledge graphs," in *Proceedings of the ISWC 2020 Demos and Industry Tracks*, ser. CEUR Workshop Proceedings, vol. 2721. Globally online: CEUR-WS.org, 2020, pp. 94–99.

[31] S. Ortona, V. V. Meduri, and P. Papotti, "Robust discovery of positive and negative rules in knowledge bases," in *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. Paris, France: IEEE Computer Society, 2018, pp. 1168–1179. [Online]. Available: https://doi.org/10.1109/ICDE.2018.00108

[32] H. J. Pandit, D. O'Sullivan, and D. Lewis, "Using ontology design patterns to define SHACL shapes," in *Proceedings of the 9th Workshop on Ontology Design and Patterns*, ser. CEUR Workshop Proceedings, vol. 2195. Monterey, USA: CEUR-WS.org, 2018, pp. 67–71.

[33] M. Pham, L. Passing, O. Erling, and P. A. Boncz, "Deriving an emergent relational schema from RDF data," in *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*. Florence, Italy: ACM, 2015, pp. 864–874. [Online]. Available: https://doi.org/10.1145/2736277.2741121

[34] E. Prud'hommeaux, J. E. L. Gayo, and H. R. Solbrig, "Shape expressions: an RDF validation and transformation language," in *Proceedings of the 10th International Conference on Semantic Systems, SEMANTiCS 2014, Leipzig, Germany, September 4-5, 2014*. Leipzig, Germany: ACM, 2014, pp. 32–40. [Online]. Available: https://doi.org/10.1145/2660517.2660523

[35] ——, "Shape expressions: an RDF validation and transformation language," in *Proceedings of the 10th International Conference on Semantic Systems, SEMANTiCS*. Leipzig, Germany: ACM, 2014, pp. 32–40.

[36] T. Quadrant, "TopBraid," https://www.topquadrant.com/products/topbraid-composer/, 2023, accessed 17th January, 2023.

[37] K. Rabbani, "Quality Shape Extraction - resources and source code," https://github.com/dkw-aau/qse, 2023, accessed 17th January, 2023.

[38] K. Rabbani, M. Lissandrini, and K. Hose, "Shacl and shex in the wild: A community survey on validating shapes generation and adoption," in *Proceedings of the ACM Web Conference 2022*. Online, Lyon, France: ACM, 2022, pp. 260–263. [Online]. Available: https://www2022.thewebconf.org/PaperFiles/65.pdf

[39] ——, "SHACL shapes for DBpedia, LUBM, YAGO-4, and WikiData published on Zenodo." https://doi.org/10.5281/zenodo.5958985, 2022.

[40] ——, "SPARQL queries used for shape extraction in QSE-Exact (query-based) approach." https://github.com/dkw-aau/qse/tree/main/src/main/resources/queries, 2023, accessed 17th January, 2023.

[41] T. Sagi, M. Lissandrini, T. B. Pedersen, and K. Hose, "A design space for RDF data representations," *VLDB J.*, vol. 31, no. 2, pp. 347–373, 2022.

[42] O. Savkovic, E. Kharlamov, and S. Lamparter, "Validation of SHACL constraints over kgs with OWL 2 QL ontologies via rewriting," in *The Semantic Web - 16th International Conference, ESWC 2019, Portorož*, ser. Lecture Notes in Computer Science, vol. 11503. Slovenia: Springer, 2019, pp. 314–329.

[43] S. Schmid, C. Henson, and T. Tran, "Using knowledge graphs to search an enterprise data lake," in *The Semantic Web: ESWC 2019 Satellite Events - ESWC*, ser. Lecture Notes in Computer Science, vol. 11762. Portorož, Slovenia: Springer, 2019, pp. 262–266. [Online]. Available: https://doi.org/10.1007/978-3-030-32327-1_46

[44] J. Sequeda and O. Lassila, "Designing and building enterprise knowledge graphs," *Synthesis Lectures on Data, Semantics, and Knowledge*, vol. 11, no. 1, pp. 1–165, 2021.

[45] B. Spahiu, A. Maurino, and M. Palmonari, "Towards improving the quality of knowledge graphs with data-driven ontology patterns and SHACL," in *Emerging Topics in Semantic Technologies - ISWC 2018 Satellite Events*, ser. Studies on the Semantic Web, vol. 36. Satellite, USA: IOS Press, 2018, pp. 103–117.

[46] T. P. Tanon, G. Weikum, and F. M. Suchanek, "YAGO 4: A reason-able knowledge base," in *The Semantic Web - 17th International Conference, ESWC*, ser. Lecture Notes in Computer Science, vol. 12123. Heraklion, Crete, Greece: Springer, 2020, pp. 583–596.

[47] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness, "Integrity constraints in OWL," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI*. Atlanta, Georgia, USA: AAAI Press, 2010.

[48] K. Thornton, H. Solbrig, G. S. Stupp, J. E. L. Gayo, D. Mietchen, E. Prud'Hommeaux, and A. Waagmeester, "Using shape expressions (shex) to share rdf data models and to guide curation with rigorous validation," in *ESWC*. Cham: Springer, 2019, pp. 606–620.

[49] D. Vrandečić and M. Krötzsch, "Wikidata: a free collaborative knowledgebase," *Communications of the ACM*, vol. 57, no. 10, pp. 78–85, 2014.

[50] W3C, "SHACL- core constraint components," https://www.w3.org/TR/shacl/#core-components, 2023, accessed 17th January, 2023.

[51] ——, "W3C: RDF Type," http://www.w3.org/1999/02/22-rdf-syntax-ns#type, 2023, accessed 17th January, 2023.

[52] WESO, "RDFShape," http://rdfshape.weso.es, 2023, accessed 17th January, 2023.

[53] WesoShaclConvert, "SHACL to ShEx converter," https://rdfshape.weso.es/shaclConvert, 2023, accessed 17th January, 2023.

[54] WikiData, "WikiData-2015," https://archive.org/details/wikidata-json-20150518, 2023, accessed 17th January, 2023.

Paper C

SHACTOR: Improving the Quality of Large-Scale Knowledge Graphs with Validating Shapes

Kashif Rabbani, Matteo Lissandrini, Katja Hose

The paper has been published in the
Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023. DOI:
10.1145/3555041.3589723

Abstract

We demonstrate SHACTOR, a system for extracting and analyzing validating shapes from very large Knowledge Graphs (KGs). Shapes represent a specific form of data patterns, akin to schemas for entities. Standard shape extraction approaches are likely to produce thousands of shapes, and some of those represent spurious constraints extracted due to the presence of erroneous data in the KG. Given a KG having tens of millions of triples and thousands of classes, SHACTOR parses the KG using our efficient and scalable shapes extraction algorithm and outputs SHACL shapes constraints. The extracted shapes are further annotated with statistical information regarding their support in the graph, which allows to identify both erroneous and missing triples in the KG. Hence, SHACTOR can be used to extract, analyze, and clean shape constraints from very large KGs. Furthermore, it enables the user to also find and correct errors by automatically generating SPARQL queries over the graph to retrieve nodes and facts that are the source of the spurious shapes and to intervene by amending the data.

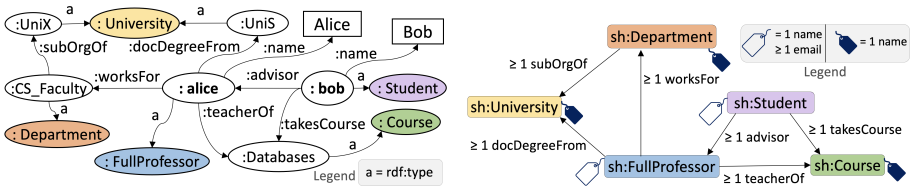
Rabbani, K., Lissandrini, M., & Hose, K. (2023, June). SHACTOR: improving the quality of large-scale knowledge graphs with validating shapes. In *Companion of the 2023 International Conference on Management of Data (pp. 151-154).* https://doi.org/10.1145/3555041.3589723
The layout has been revised.

1 Introduction

Knowledge Graphs (KGs) are in widespread use both within companies and on the Web [5, 9], thanks to their ability to represent a wide range of information in different domains. DBpedia (`dbpedia.org`) and WikiData (`wikidata.org`) are examples of large public KGs, where data is stored using the Resource Description Framework (RDF) [1], which by design allows modeling data without a schema definition (unlike, for instance, the relational model). In RDF, data is modeled as nodes representing entities and data values, while edges represent relationships and attributes (see the example in Figure C.1a).

Nonetheless, as more and more data is accrued within KGs, practical applications impose further demands regarding quality assessment and validation. To provide a way to validate the contents of a KG, shape constraint languages, namely SHACL [4] and ShEx [6], have been proposed as ways to define and enforce constraints using so-called *validating shapes*. Validating shapes allow to define a partial schema for the entities end relationships contained in a KG and overcome some of the limitations of the RDF Schema specification while being easier to use than OWL ontologies. Shapes can be used to express that an entity of type Student needs to have a name, an advisor, and should be enrolled in some courses; and that these attributes should be instances of type string, FullProfessor, and Course, respectively (see Figure C.1b for a simplified depiction of some validating shapes describing the relationships between some entity types). Thanks to their simplicity and expressivity, shape constraint languages have attracted increasing interest, and SHACL became a W3C standard in 2017 [4].

Validating shapes have attracted substantial attention in the past few years, and recently, we conducted a survey [7] to analyze the extraction and adoption of validating shapes in industry and academia. Data scientists are faced with the challenge to *craft a set of validating shapes for already existing KGs* that they are working with and then use those to clean such datasets, i.e., a *post-hoc* validation. Hence, we face the need to develop semi-automatic



(a) RDF Graph **(b)** Validating Shapes

Fig. C.1: An example RDF Graph and Validating Shapes

methods to help users generate shapes for large existing KGs. These are usually the shape extraction approaches [2, 3]. Due to well-known issues in data quality and the heterogeneity of existing KGs, spuriousness poses important challenges to automatic shape extraction methods. For example, in DBpedia, a few entities representing musical bands are wrongly assigned to dbo:City class. As a consequence, when shapes are extracted from its instance data using approaches that do not analyze the support of the shapes, the resulting node shape for dbo:City specifies that cities are allowed to have dbo:genre and dbo:formerBandMember properties. We call these *spurious shapes*.

Existing shape extraction approaches produce many spurious shapes when dealing with erroneous triples or incomplete data [8]. Thus, extraction approaches will generate tens of thousands of shape constraints due to the effect of *spuriousness*. In these situations, it becomes *unmanageable for domain experts to manually analyze and clean* the resulting validating shapes. Finally, *existing methods are not scalable*, i.e., unable to extract shapes from very large KGs (especially on commodity machines). Therefore, we proposed a solution called *Quality Shapes Extraction (QSE)* [8] to tackle both the limitations of *scalability* and *spuriousness* in existing shapes extraction approaches. QSE extracts validating shapes from very large graphs on a commodity machine (it takes only 3 minutes on DBPedia with just 16 GB of RAM) and also provides information about the reliability of the extracted shape constraints by computing their *confidence* and *support*. Hence, QSE identifies those shapes that are the most informative and distinguishes those that are indeed affected by incomplete or incorrect data.

Contributions. In this work, we show how the shapes extracted with QSE in combination with the information about their confidence and support enable a wide range of data profiling and cleaning functionalities, beyond simple validation. We thus propose SHACTOR (for SHapes extrACTOR), a tool that data scientists can use *to speed up the end-to-end KG cleaning process* by (1) automatically extracting shapes, (2) *helping to evaluate their quality*, (3) providing *important structural profiling information*, and (4) allowing *to find errors and missing data* in the given KG as well as correct such issues by *automatically generating and executing SPARQL queries*. SHACTOR uses QSE [8] to filter shapes by various confidence and support thresholds to identify reliable shapes. These shapes provide essential information on the structure and content of the KG as well as on possible data quality issues. SHACTOR then highlights the spurious shapes and automatically generates SPARQL queries to extract the erroneous or incomplete data generating such shapes. The system allows for removing the erroneous triples interactively and then correcting the generated shapes. Thus, the tool will lead to a valid set of shapes, which can be used to maintain the quality of the given KG in the future.

Demo Video and Source Code. The demonstration video and source

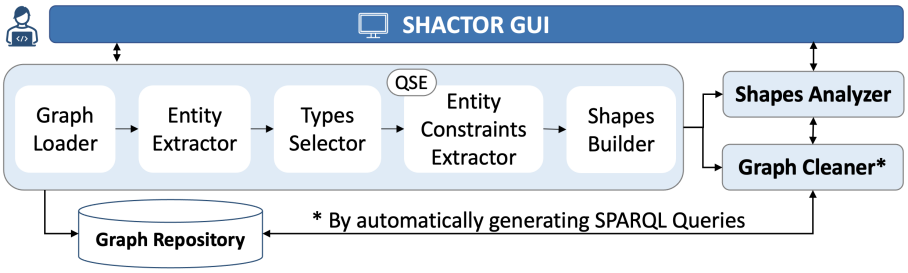


Fig. C.2: SHACTOR Architecture

code of SHACTOR is available on our website and GitHub[1].

2 SHACTOR

SHACTOR provides a graphical user interface for interacting with our QSE algorithm [8] and provides a wide range of new functionalities. Given a KG in RDF, SHACTOR uses QSE to extract a full set of validating shapes and then allows the user to explore and analyze this output. Hence, SHACTOR consists of three main phases, (1) shapes extraction with support and confidence, (2) shapes analysis, and (3) KG cleaning. Figure C.2 shows SHACTOR's architecture.

Background and Problem. An RDF knowledge graph models entities and their relationships in the form of <subject, predicate, object> triples. Given pairwise disjoint sets of IRIs $\mathcal{I}$, blank nodes $\mathcal{B}$, and literals $\mathcal{L}$, an RDF Graph $\mathcal{G}{:}\langle N, E\rangle$ is a graph with a finite set of nodes $N{\subset}(\mathcal{I}{\cup}\mathcal{B}{\cup}\mathcal{L})$ and a finite set of edges $E{\subset}\{\langle s,p,o\rangle{\in}(\mathcal{I}{\cup}\mathcal{B}) \times \mathcal{I} \times (\mathcal{I}{\cup}\mathcal{B}{\cup}\mathcal{L})\}$. See a sample $\mathcal{G}$ in Figure C.1a, where oval and rectangular shapes represent IRIs and literal nodes, respectively. The standard query language for RDF data is SPARQL. A SPARQL query Q consists of a set of triple patterns along the conditions that have to be met in order for data in $\mathcal{G}$ to contribute to the result.

The SHACL shapes of $\mathcal{G}$ are defined as set of node shapes $\mathcal{S}{:}\{\langle s, \tau_s, \Phi_s\rangle, ...\}$, where s is the shape *name*, $\tau_s{\in}\mathcal{C}$ is the *target class*, and Φ_s is a set of property shapes of the form $\phi_s{:}\langle \tau_\mathsf{p}, \mathsf{T_p}, \mathsf{C_p}\rangle$, where $\tau_\mathsf{p}{\in}\mathcal{P}$ is called the target property, $\mathsf{T_p}{\subset}\mathcal{I}$ contains either an IRI defining a *literal type*, e.g., xsd:string, or a set of IRIs – called *class type constraint*, and $\mathsf{C_p}$ is a pair $(n,m) \in \mathbb{N} \times (\mathbb{N}{\cup}\{\infty\})$. $n{\leq}m$ – called min and max *cardinality constraints*. For example, the node shape for :Student in $\mathcal{G}$ (Figure C.1a) is $\{s$:studentNodeShape, τ_s : Student, $\Phi_s{:}\{\phi_{\text{:name}}, \phi_{\text{:takesCourse}}, \phi_{\text{:advisor}}\}\}$ where

[1] https://relweb.cs.aau.dk/qse/shactor/, https://github.com/dkw-aau/demo-shactor

property shape $\phi_{:\mathsf{name}}$ has constraints $\langle$:name, xsd:string, $(1, \infty)\rangle$, i.e., every node of type Student should have at least one name of type string (see Figure C.1b).

In QSE, we introduced the notion of *support* and *confidence* for shape constraints to study the reliability of extracted shapes and tackle the issue of spuriousness. These concepts are inspired by the well-known theory developed for the task of frequent pattern mining. The *support* ω of a constraint measures how many entities are conforming to a specific constraint $\phi_s{:}\langle\tau_\mathsf{p}, \mathsf{T}_\mathsf{p}, \mathsf{C}_\mathsf{p}\rangle \in \Phi_s$ of a shape $\langle s, \tau_s, \Phi_s\rangle \in \mathcal{S}$ appearing in the data graph $\mathcal{G}$. Similarly, the *confidence* ε of a constraint ϕ_s measures the ratio between how many entities conform to ϕ_s and the total number of entities that are instances of the target class of the shape s.

Given the need to improve the quality of the data within an existing KG, SHACTOR *addresses the problem of helping users to produce high-quality validating shapes and use them to analyze and correct the data quality issues present in the graph*. Hence, to extract shapes from a large existing graph $\mathcal{G}$ while tackling the effects of spuriousness and helping the user to *analyze* $\mathcal{S}$ and *clean* $\mathcal{G}$, the SHACTOR system takes a knowledge graph $\mathcal{G}$, a threshold ω for support, and ε for confidence and produces all shapes after distinguishing for which subset of node shapes $\langle s, \tau_s, \Phi_s\rangle{\in}\mathcal{S}$ it holds that $\mathrm{supp}(s){>}\omega$ and for which property shapes $\phi_s{:}\langle\tau_\mathsf{p}, \mathsf{T}_\mathsf{p}, \mathsf{C}_\mathsf{p}\rangle{\in}\Phi_s$, $\mathrm{supp}(\phi_s){>}\omega$ and $\mathrm{conf}(\phi_s){>}\varepsilon$. Moreover, for each $\Phi_s \in \mathcal{S}$, SHACTOR generates a SPARQL query Q to fetch either triples that conform to some $\phi_s \in \Phi_s$ or triples that fail to conform to it, allowing to inspect specific data quality issues. Thus, SHACTOR is a system that *takes full advantage of existing standards* in terms of data formats, query languages, and validation constraints in the realm of KGs, while also ensuring ease of use and scalability.

Shapes Extraction and Analysis. SHACTOR parses $\mathcal{G}$ (available as SPARQL endpoint or from file) to extract entities and their constraints and computes support and confidence for each constraint. To focus on the specific subset of the graph, it also allows selecting a custom subset of types in the KG, where the user can select one or more classes to focus only on entities of the specific class, e.g., :Student or :FullProfessor class in Figure C.1a. In the second step, it asks to input thresholds for support ω and confidence ε. Given this information, it will use the QSE algorithm to parse $\mathcal{G}$ and produce all shapes that satisfy the support and confidence thresholds while also producing shape constraints that fail to meet these conditions (See for instance ① in Figure C.3). The data scientists can further dynamically filter the produced shapes by providing more restrictive values for ω and ε. They can further fine-tune the pruning thresholds by using statistical information like the frequency of each class. Note that while a spurious shape does not usually represent a valid constraint to be enforced, it usually signals the presence

of erroneous data in the graph. Hence, on the one hand, SHACTOR provides information w.r.t. the contents and structure of the KG, helping in identifying a set of reliable validating shapes to be enforced, while, on the other hand, by looking at the shapes with low support and confidence, it also helps in understanding which portions of the data may be particularly problematic. *To the best of our knowledge, SHACTOR is the first tool that considers both support and confidence to identify errors in a KG.*

Therefore, to further inspect the shapes produced in this way, SHACTOR implements several useful features. First, it displays interactive charts that show which percentage of shapes are above and below each threshold (see ② in Figure C.3). Furthermore, in the list of node shapes, i.e., for each node shape s in $\mathcal{S}$, it shows the support ω_s of s along with the quality of the data behind each of them in terms of the number of property shapes specific for that node shape (i.e., $|\Phi_s|$ of s) that are above the two thresholds ω and ε (see ③ in Figure C.3). Node shapes with many property shapes with low support and confidence signal the presence of noisy or incomplete data for entities of that type.

Moreover, for each property shape $\phi_s \in \Phi_s$ of a given node shape $s \in \mathcal{S}$, it shows which predicate and type constraints it involves (the values of $\langle \tau_\mathsf{p}, \mathsf{T}_\mathsf{p}, \mathsf{C}_\mathsf{p} \rangle$) along with its specific support ω_{ϕ_s} and quality indicator of each property shape ϕ_s computed by visually showing the confidence of the shape and highlighting those property shapes whose confidence is below the user-provided threshold (see ④ in Figure C.3). These quality indicators and highlights based on support and confidence of each node shape and its property shapes help the user find the spurious shape constraints in $\mathcal{S}$.

Improving Data Quality. The result of validating a KG using validating shapes is usually a validation report that lists all entities and triples that violate the given set of constraints expressed by the shapes. Nonetheless, as we have seen, when using an automatic shape extraction tool, the data scientist needs to decide which subset of shapes to enforce among those extracted by the tool. SHACTOR helps data scientists by automatically generating queries to retrieve the entities and triples that caused a given shape to be extracted from the data. In a sense, it allows to inspect the *provenance* of a shape. This helps the user both in deciding which shapes to be used for validation as well as to identify either erroneous triples or missing information when inspecting spurious shapes.

Specifically, for a given node shape $s \in \mathcal{S}$, SHACTOR can build SPARQL queries for each property shape $\phi_s \in \Phi_s$ using the property path τ_p, class type constraint T_p of ϕ_s, and target class τ_s. Hence, given the node shape s with property shapes Φ_s, SHACTOR automatically builds a SPARQL query to retrieve all the triples having property path τ_p and class type τ_s for a given $\phi_s \in \Phi_s$, e.g., to inspect which entities of type dbo:City in DBpedia have the

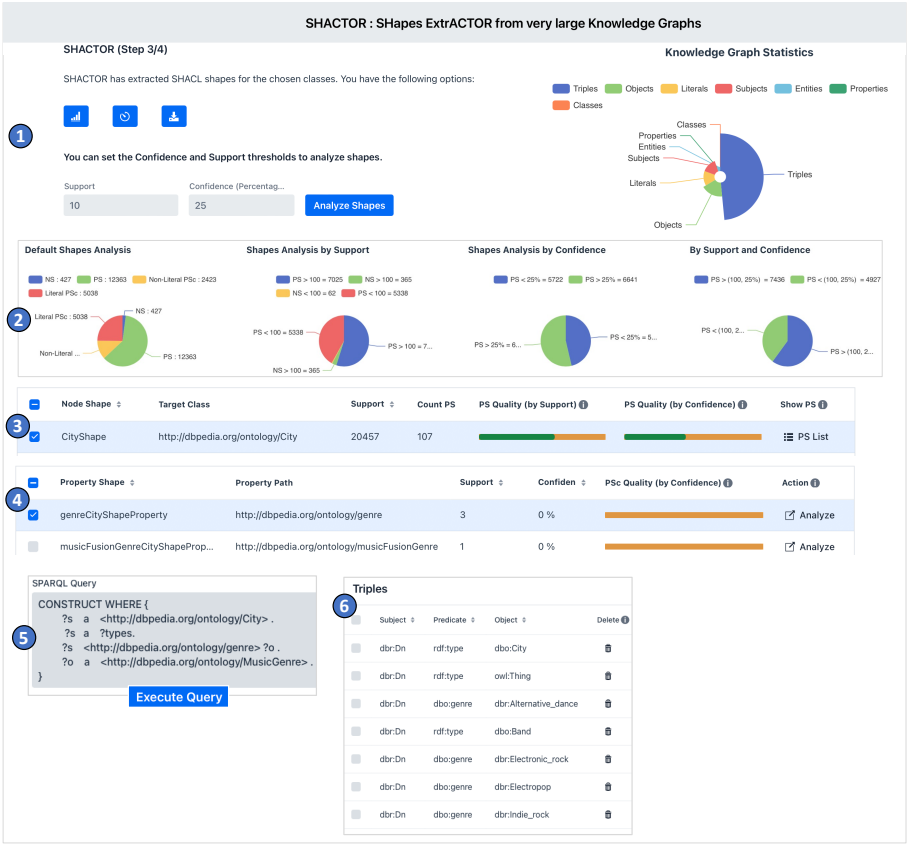


Fig. C.3: Portions of SHACTOR GUI showing a sample of the analysis of the shapes for DBpedia

genre attribute (see ⑤ in Figure C.3). SHACTOR can then directly execute the query and return the corresponding list of triples (see ⑥ in Figure C.3). Further, SHACTOR provides the option to generate the queries to delete selected triples, thus allowing to automatically delete erroneous triples on the spot, as well as to generate INSERT queries to amend the missing information in the graph.

3 Demonstration Scenario

SHACTOR demonstrates the power and versatility of a tool that effectively exploits the information carried by shapes extracted and annotated with statistical data. We show how data scientists can use SHACTOR to speed up the process of KG cleaning by automatically extracting shapes and evaluating the quality of extracted shapes along with their provenance. First, the demonstra-

tion guides the participants through different extraction phases (not shown in the figure). We use a full snapshot of DBpedia from 2021, which contains 52 M triples, 15 M literals, 5 M typed entities, 1.3 K properties, and 427 classes. Moreover, participants will also be able to analyze shapes from a full snapshot of WikiData (having more than 1.9 billion triples) that we have extracted in advance. SHACTOR can extract shapes on a commodity machine also for WikiData [8], but while for DBpedia, it takes only a few minutes, for WikiData, it takes too long for a live demonstration of the extraction.

Configuration. At first, we present various options to provide a KG as input, e.g., the user can upload a KG file or point to a SPARQL endpoint. *The audience is also welcome to bring their own KG to be analyzed.* Then, the user selects the target sub-graph (from the list of extracted classes); after that, SHACTOR starts the shapes extraction step and the user is directed to the main analysis interface (Figure C.3). SHACTOR also supports uploading already extracted shapes.

Shapes Analysis. The user provides support and confidence as pruning thresholds to analyze the extracted shapes. Hence, SHACTOR applies the input pruning thresholds and display an overview in the form of pie charts (see ① and ② in Figure C.3). These charts show the portions of node and property shapes that are below or above the provided pruning thresholds. Furthermore, they help the user decide the optimal values for the pruning thresholds. In our DBpedia example, we see the quality indicators for two node shapes (shown in ③ of Figure C.3). Then, we show an example node shape called :CityShape having target class dbo:City and explore its property shapes (shown in ④ of Figure C.3). The list of shape constraints can be sorted by increasing or decreasing support so that the user can select the desired set of node or property shapes. The selected shapes can also be downloaded before or after applying pruning thresholds to be used with a shape validation tool.

Finding Errors in the KG. SHACTOR helps users find spurious shape constraints by highlighting constraints having support or confidence less than the provided input pruning thresholds. Consider a user exploring the property shapes of :CityShape having the lowest support and confidence, the user will find out the :genre property shape in the set of property shapes for :CityShape with support of 3. SHACTOR, by default, highlights the property constraints below the pruning thresholds in red color. Hence, this will help the user easily find such spurious property shapes. This shape signals the presence of some erroneous triple. To improve the quality of data in the KG, the user may wish to locate and remove the triples involved in such a shape. Thus, with the click of a button, SHACTOR can generate the SPARQL query from the interface for retrieving the triples involved in the :genreCityShapeProperty (shown in ⑤), and upon execution, the retrieved

triples will be fetched from the KG and displayed on the interface (shown in ⑥). The list of retrieved triples will show the entities which are wrongly classified, e.g., dbo:Heris is classified as dbo:City and dbo:Song. Now the user can go through the list of triples and take appropriate action to improve the quality of data in the KG. In particular, the user can select a subset of triples they wish to delete, and SHACTOR will generate a corresponding DELETE query. For instance, the triples assigning the wrong type to the entities dbr:Dn, dbr:Heris, and dbr:agar.

Finding Missing information in the KG. SHACTOR allows the user to explore the property shapes in a separate dashboard (not shown in Figure C.3). Here, for each property shape, it identifies also *object IRIs that are missing types* as well as *entities that are missing properties*. Based on these, SHACTOR suggests INSERT queries to add the missing information. For example, it suggests to add a value for the dbo:capital property for those entities of type dbo:country that are missing a capital.

4 CONCLUSION

In this demo paper, we present SHACTOR, a system to support end-to-end profiling and cleaning of large-scale KGs using validating shapes automatically extracted from the graph and annotated with statistical information thanks to our scalable QSE algorithm. Further, this demo shows the versatility and effectiveness of utilizing shapes as easy-to-extract and easy-to-use tools to identify and correct data quality issues in existing KGs. While our tool allows to analyze shapes in isolation, in the future, we plan to add a feature to visualize the shapes-based schema graph to analyze the taxonomy of the shapes.

References

[1] W. Consortium, "RDF 1.1," https://w3.org/RDF/, 2014.

[2] D. Fernandez-Álvarez, J. E. Labra-Gayo, and D. Gayo-Avello, "Automatic extraction of shapes using shexer," *KBS*, vol. 238, p. 107975, 2022.

[3] A. Keely, "SHACLGEN," https://pypi.org/project/shaclgen/, 2022.

[4] H. Knublauch and D. Kontokostas, "Shapes constraint language (shacl)," *W3C Candidate Recommendation*, vol. 11, no. 8, 2017.

[5] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor, "Industry-scale knowledge graphs: lessons and challenges," *ACM*, vol. 62, no. 8, pp. 36–43, 2019.

[6] E. Prud'hommeaux, J. E. L. Gayo, and H. R. Solbrig, "Shape expressions: an RDF validation and transformation language," in *ICSS*, 2014, pp. 32–40.

[7] K. Rabbani, M. Lissandrini, and K. Hose, "Shacl and shex in the wild: A community survey on validating shapes generation and adoption," in *TheWebConf-2022*, 2022, pp. 260–263.

[8] ——, "Extraction of validating shapes from very large knowledge graphs," *PVLDB*, vol. 16, no. 5, pp. 1023–1032, 2023.

[9] J. Sequeda and O. Lassila, "Designing and building enterprise knowledge graphs," *Synthesis Lectures on Data, Semantics, and Knowledge*, vol. 11, no. 1, pp. 1–165, 2021.

Paper D

Optimizing SPARQL Queries using Shape Statistics

Kashif Rabbani, Matteo Lissandrini, Katja Hose

The paper has been published in the
Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021. DOI: 10.5441/002/EDBT.2021.59

Abstract

With the growing popularity of storing data in native RDF, we witness more and more diverse use cases with complex SPARQL queries. As a consequence, query optimization – and in particular cardinality estimation and join ordering – becomes even more crucial. Classical methods exploit global statistics covering the entire RDF graph as a whole, which naturally fails to correctly capture correlations that are very common in RDF datasets, which then leads to erroneous cardinality estimations and suboptimal query execution plans. The alternative of trying to capture correlations in a fine-granular manner, on the other hand, results in very costly preprocessing steps to create these statistics. Hence, in this paper we propose shapes statistics*, which extend the recent SHACL standard with statistic information to capture the correlation between classes and properties. Our extensive experiments on synthetic and real data show that shapes statistics can be generated and managed with only little overhead without disadvantages in query runtime while leading to noticeable improvements in cardinality estimation.*

Rabbani, K., Lissandrini, M., & Hose, K. (2021). Optimizing SPARQL queries using shape statistics. In *Advances in Database Technology-24th International Conference on Extending Database Technology, EDBT 2021 (pp. 505-510)*
http://dx.doi.org/10.5441/002/edbt.2021.59
The layout has been revised.

1 Introduction

Driven by diverse movements, such as Linked Open Government Data, Open Street Map, DBpedia [3], and YAGO [21], more and more data is being published in RDF [7] capturing a multitude of diverse information. Along with the growing popularity, increasingly complex queries formulated in SPARQL [6] are being executed over such data to answer business and research questions. Query logs of the public DBpedia SPARQL endpoint, for instance, contain SPARQL queries with up to 10 joins [4] and analytic queries in the biomedical field can involve more than 50 joins per query [9]. Therefore, the need for high-performance SPARQL query processing is now more pressing than ever.

Existing approaches for query optimization in RDF stores often adapt techniques from relational databases modeling an RDF dataset as a single large table with three column [5, 16] (one column for each of the components of an RDF triple: subject, predicate, and object). Nevertheless, accurate cardinality estimation is at the heart of any query optimizer that does not rely on heuristics but instead uses a cost model to find the best query execution plan for a given query. Cardinality estimation then relies on the availability of statistics describing the characteristics of the data to estimate the sizes of intermediate results produced while query execution. However, general statistics typically result in highly imprecise estimations since they are mostly gathered on the RDF graph as a whole, in contrast to the relational case where it is possible to create such statistics with higher precision since data is separated into multiple tables [15]. Furthermore, assuming independence when joining parts of SPARQL queries (triple patterns) leads to erroneous estimations [9] as co-occurrences of certain predicates are highly correlated [19].

Hence, exploiting more fine-grained statistics capturing correlations among RDF triples leads to more accurate join cardinality estimations [19]. However, creating such statistics comes at the price of a very time and resource-intensive preprocessing step. On the other hand, the alternative of online, query-dependent, sampling [20] results in overheads during query optimization. Instead, what we propose in this paper is to better exploit the information that is often provided along with an RDF dataset: SHACL (Shapes Constraint Language) [14] constraints, which is a recent standard for validating RDF datasets that are becoming more and more popular. SHACL defines so-called shapes describing the relationships between entities of a specific class, their properties, and their connections to other classes of entities. Although they are currently only used for validation purposes, we show in this paper that by slightly extending them with basic statistics, they can also be exploited for join cardinality estimation.

In summary, this paper makes the following contributions. First, we extend the SHACL definition to capture statistical information to replace the need for creating complex (and expensive) statistics over RDF datasets. To the best of our knowledge, this is the first proposal of this kind. Second, we introduce an algorithm to enhance SHACL shapes with statistical information and to exploit these statistics for join cardinality estimation and query optimization. Third, we study the impact of our approach using both synthetic (LUBM [10], WatDiv [2]) and real (YAGO-4 [21]) datasets, demonstrating that shapes statistics can provide higher precision for query optimization with only a little overhead.

This paper is structured as follows. While Sections 2 and 3 discuss related work and introduce preliminaries, Section 4 formally defines the problem. Section 5 then describes our proposed extension of the SHACL standards, and Section 6 presents techniques to exploit the additional information for cardinality estimation and query optimization. Section 7 discusses the results of our extensive experimental study, and Section 8 concludes the paper with an outlook to future work.

2 Related Work

Cardinality estimation has been studied extensively in the context of relational databases [20]. For SPARQL queries, existing techniques adapt relational approaches [13, 24] and focus mostly on specific type of queries [19]. Usually, these approaches construct different kinds of single or multidimensional synopses over databases that can be used to estimate cardinalities [23]. While algorithms designed to generate synopsis for unlabelled graphs are not applicable here (as the edges in RDF graphs are labeled), consequently approaches to generate RDF summaries either produce very large summaries [23], have very high computational complexities, or they are unable to preserve the RDF schema while constructing the summaries [23]. Therefore, the most promising approaches aim at using statistics computed directly from edge label frequencies. In particular, RDF-3X proposes a histogram-based technique for cardinality estimation based on edge label frequencies. This technique was later extended by exploiting the statistical information of Characteristic Sets [19], which compute frequencies of sets of predicates sharing the same subject to estimate the cardinalities. This approach shows high performance for star-shaped queries while it suffers from significant underestimation due to the independence assumption in the general case [20]. This approach was extended as Characteristic Pairs [18] to overcome this limitation, but it could only support multi-chain star queries. Moreover, extracting Characteristic Sets from large heterogeneous graphs is computationally expensive. SumRDF [23] is another cardinality estimation approach based on

a graph summarization. It fails to handle large queries due to a prohibitive computation cost, and it is costly to construct such summaries over large RDF graphs [20].

A recent benchmark, G-CARE [20], analyzed the performance of existing cardinality estimation techniques for subgraph matching. This analysis revealed that the techniques based on sampling and designed for online aggregation outperform the cardinality estimation techniques for RDF graphs. *This calls for a more in-depth study on how to perform cardinality estimation for SPARQL query optimization appropriately*.

In a recent work, Shape Expressions (ShEx) [22] have been used to reorder triple patterns to enable SPARQL query optimization [1], i.e., it estimates an order of execution for the triple patterns based on some heuristic inference on which triples are more selective. For instance, if a shape definition says that every instructor has one or more courses, but every course has exactly one instructor, it infers that the cardinality of courses is at least the same as the cardinality of instructors and probably larger. Hence, this optimization procedure is not based on actual data.

Therefore, contrary to existing works, we aim at exploiting fine-grained statistics based on shapes to produce more precise cardinality estimations for query planning. This will allow us to overcome the limitations of existing methods that only use the global-statistics [11]. To this end, instead of creating large expensive summaries and characteristic sets over the RDF graphs to estimate the cardinalities, we exploit SHACL shapes constraints (which are as expressive as ShEx [22]) and annotate the *Node* and *Property Shapes* with the statistics of the input RDF graph. Compared to other solutions, it requires a lightweight preprocessing and retains the structure of original RDF and SHACL shapes graphs. Moreover, *this allow us to study more closely the effect of more fine-grained statistics, and more accurate cardinality estimation for the task of SPARQL query optimization*.

3 Preliminaries

RDF Graphs: RDF graphs model entities and their relationships in the form of triples consisting of SPO <*subjects, predicates, objects*>. We present a simplified example of an RDF graph G based on the LUBM [10] dataset in Figure D.1, where oval and rectangular shapes represent IRIs and literal nodes, respectively. An RDF graph is formally defined as:

Definition D.1 (RDF Graph)
Given pairwise disjoint sets of IRIs I, blank nodes B, and literals L, an RDF Graph G is a finite set of RDF triples $\langle s, p, o \rangle \in (I \cup B) \times I \times (I \cup B \cup L)$.

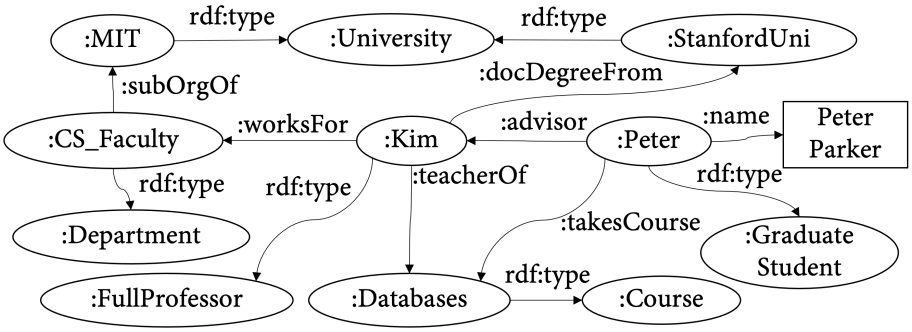


Fig. D.1: An RDF Graph G

```
SELECT * WHERE {
    ?X :degreeFrom      ?U.
    ?X rdf:type         :GraduateStudent.
    ?X :advisor         ?A .
    ?A rdf:type         :FullProfessor.
    ?A :teacherOf       ?C.
    ?A :name            ?N.
    ?C rdf:type         :GraduateCourse.
    ?Y :takesCourse     ?C.
    ?Y rdf:type         :GraduateStudent.
}
```

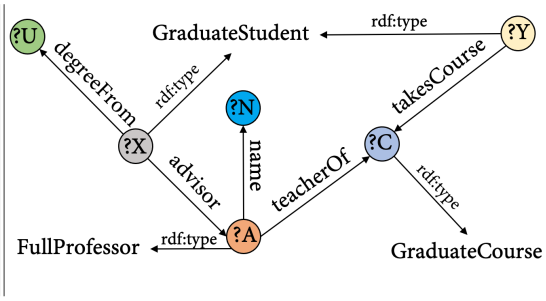


Fig. D.2: Query Q and its Graph Q_G

SPARQL: SPARQL [6] is a standard query language for RDF. A SPARQL query consists of a finite set of triple patterns (known as basic graph pattern, BGP) and some conditions that have to be met in order for data to be selected and returned from an RDF graph. Each SPO position in a triple pattern can be concrete (i.e., bound) or a variable (i.e., unbound). The variable names in a SPARQL query are prefixed by a '?' symbol, e.g., `?X`. To answer a BGP, we require *a mapping between variables to values in an RDF graph*, all the resulting triples existing in the RDF graph obtained by replacing the variables with values are answers to the BGP. Figure D.2 shows an example SPARQL query (Q) and its query graph Q_G on the graph of Figure D.1. A BGP is defined as:

Definition D.2 (BGP)
Given a set of IRIs I, literals L, and variables V, a BGP is defined as $T \subseteq (I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$, whose elements are called triple patterns.

Shapes Graphs: Several schema languages have been proposed for RDF in the past, where the most common are RDF Schema (RDFS[1]) and OWL [17].

[1] https://www.w3.org/TR/rdf-schema/

RDFS is primarily used to infer implicit facts, and OWL is an extension of RDF and RDFS to represent ontologies. The declarative Shapes Constraint Language (SHACL) [14] became a W3C standard recently. SHACL schema provides high-level information about the structure and contents of an RDF graph. It allows to define and validate structural constraints over RDF graphs. SHACL models the data in two components: the *data graph* and the *shape graph*. The *data graph* contains the actual data to be validated, while the *shape graph* contains the constraints against which resources in the *data graph* are validated. These constraints are modeled as node and property shapes, which consist of attributes encoding the constraints. The node shapes constraints are applicable on nodes that are instances of a specific type in the *data graph* while the property shapes constraints are applicable to predicates associated with nodes of specific types. We define a SHACL shapes graph as follows:

Definition D.3 (SHACL Shapes Graph)
A SHACL shapes graph G_{sh} is an RDF graph describing a set of node shapes S and a set of property shapes P, such that $target_S : S \mapsto I$ and $target_P : P \mapsto I$ are injective functions mapping each node shape $s_i \in S$ and each property shape $p_i \in P$ to the IRI of a target class and a target predicate in G respectively, and $\phi : S \mapsto 2^P$ is a surjective function assigning to each node shape s_i a subset $P_i \subseteq P$ of property shapes.

For example in Figure D.3, node shape constraints are applicable on node `ub:GraduateStudent` and its property shapes constraints are applicable on predicates like `takesCourse`, and `advisor`. This information is declared with attributes *sh: targetClass* for node shapes and *sh:path* for property shapes. Note that the attributes in the dark shaded boxes are part of our extension of the SHACL definition, explained in Section 5.

The Shapes Expression (ShEx [22]) language also serves a similar purpose as SHACL to validate RDF graphs. Nonetheless, the two formulations diverge mostly at the syntactic level [12], and our approach can be extended to work using ShEx or other constraints languages as well without the loss of generality.

4 Problem Formulation

Given an input query Q, a query optimizer has the goal to find a query plan expected to answer Q in the minimum amount of time [15]. Constructing a SPARQL query plan includes finding a join ordering between triple patterns of its BGPs. In this paper, we focus on the join ordering of BGPs defined as follows:

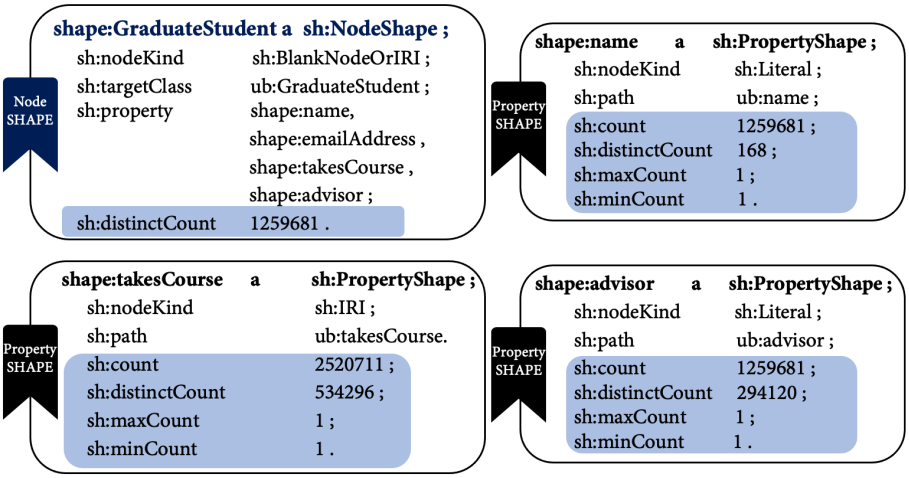


Fig. D.3: SHACL Shapes Graph

Definition D.4 (Join Ordering)
Given a set of triple patterns $T=\{tp_1, tp_2, ..., tp_n\} \subseteq (I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$, the join order $\mathcal{O}$ for BGPs is defined as a total ordering $\mathcal{O}$ of T so that for every $t_i, t_j \in T$ either $t_i \prec_{\mathcal{O}} t_j$ or $t_j \prec_{\mathcal{O}} t_i$.

To find an optimal plan, a query optimizer needs to explore the search space of semantically equivalent join orders and choose the optimal (cheapest) plan according to some cost function. It is crucial to accurately estimate the join cardinality between triple patterns of a given query to construct a query plan with an efficient join ordering [9]. In line with the related work [20], we neglect other cost factors and focus on join cardinality as the most dominant cost factor to find a join ordering. We formally define the problem of estimating join cardinalities as follows:

Problem 2 (Join Cardinality Estimation)
Given a set of triple patterns $T=\{tp_1, tp_2, ..., tp_n\}$, apply a cardinality estimation function $\bar{J} : T \times T \mapsto \mathbb{N}$ such that for every pair of triple patterns $(tp_i, tp_j) \in T$, $\bar{J}(tp_i, tp_j) \approx |tp_i \bowtie tp_j|$.

We extend the above estimation problem also to the case of joining a triple pattern with the intermediate results of prior join operations, e.g., to estimate the total cardinality $\bar{J}((tp_i \bowtie tp_j), tp_k) \approx |(tp_i \bowtie tp_j) \bowtie tp_k|$. Then, given such estimates, an optimal query plan minimizes the total number of operations to compute, i.e., the execution costs $Cost(T, \mathcal{O})$ of the order $\mathcal{O}$ for the set T. In practice, this total join cost is obtained by summing up the intermediate cardinalities of each join operation in their respective join order. Hence, we formalize the problem of join order optimization as follows:

Problem 3 (Join Order Optimization)
Given a set of triple patterns $T=\{tp_1, tp_2, ..., tp_n\}$ and a join cardinality estimation function $\bar{J}$, find the join order $\mathcal{O}$ obtained as $\arg\min_{\mathcal{O}} Cost(T, \mathcal{O})$.

5 Extending SHACL with Statistics

To compute more accurate join cardinality estimations (Problem 2), we capture the correlations between RDF triples by extending SHACL's node and property shapes with fine-grained statistics of the RDF graph. We denote these statistics as *shapes statistics*. These include the total triple count (*sh:count*), minimum (*sh:minCount*) and maximum (*sh:maxCount*) number of triples for each instance, and the number of distinct objects for property instantiations (*sh:distinctCount*). The attributes shown in the dark shaded boxes in Figure D.3 are the annotated statistical attributes of their respective node and property shapes. These statistics are computed by executing analytical SPARQL queries over the RDF graph. For instance, to compute the number of instances of *GraduateStudent* in the dataset, i.e, the value of attribute *sh:count* of node shape *GraduateStudent*, the annotator issues the SPARQL query: `SELECT COUNT(*) WHERE {?x a ub:GraduateStudent}`.

Along with *shapes statistics*, we also define *global statistics* by extending VOID[2] statistics with more precise statistics of RDF properties, i.e., the distinct subject count (DSC) and distinct object count (DOC) of each property of the RDF graph.

6 Query Planning

In this section, we present our approach to exploit global and shapes statistics to obtain more accurate join cardinality estimates (Problem 2). These estimates, in turn, are used for join order optimization (Problem 3).

6.1 Cardinality Estimation of Triple Patterns

A SPARQL query contains joins between multiple triple patterns. Hence, the first step is to estimate how many triples match every triple pattern individually. We exploit the statistical information contained in the extended SHACL shapes graph (Section 5) to obtain this estimate. Hence, for each triple pattern, we obtain their corresponding node or property shapes using the values of the *sh:targetClass* and *sh:path* attributes.

First, all triples of the type <*?x, a, [Class]*> (i.e., instances with *rdf:type [Class]*) are mapped to the node shape having that class as the value of

[2]Vocabulary of Interlinked Datasets: https://www.w3.org/TR/void/

the attribute *sh:targetClass*. Then, triples having variable *?x* as a subject are also assigned to that node shape. The triple predicate determines instead its corresponding candidate property shapes, i.e., those with a matching value for *sh:path*. For example, given triples tp_1=<*?x, rdf:type, ub:GraduateStudent*> and tp_2=<*?x, ub:name, ?n*>, the subject *?x* is assigned to node *shape:GraduateStudent*, while the predicate in tp_2 matches *shape: name* (Figure D.3, top left and top right).

Once the candidate shapes for all the triple patterns are identified, their statistical information combined with the distinct subject and object count (DSC & DOC) from the *global statistics* are used in combination with the formulas shown in Table D.1 to compute their expected cardinality. These formulas, inspired by a previous work [11], cover all possible types of triple patterns. The term c_X in the formulas denotes the count of X in the RDF graph; $c_{triples}$ denotes the count of all triples and $c_{objects}$ the count of all objects. Similarly, c_{X_Y} represents the count of X having Y. This can be used, for instance, to derive that there are $\sim$ 85K triples matching <*?x, rdf:type, ub:FullProfessor*> (Table D.2a). While both global and shapes statistics can be used to estimate the cardinality of triple patterns using these formulas, they can lead to different estimated cardinalities. When the query does not contain any type-defined triple, only global statistics are used.

Triple Pattern	Cardinality	Triple Pattern	Cardinality
?s ?p obj	$\frac{c_{triples}}{c_{objects}}$	?s ?p ?o	$c_{triples}$
subj ?p obj	$\frac{c_{triples}}{c_{distSubj} \times c_{distObj}}$	subj ?p ?o	$\frac{c_{triples}}{c_{distSubj}}$
?s pred obj	$\frac{c_{pred}}{c_{pred_{obj}}}$	?s pred ?o	c_{pred}
subj pred obj	$\frac{c_{pred}}{c_{distSubj} \times c_{distObj}}$	sub pred ?o	$\frac{c_{pred}}{c_{pred_{sub}}}$
?s rdf:type obj	$c_{entities_{rdf:type_{obj}}}$	?s rdf:type ?o	$c_{rdf:type}$
subj rdf:type obj	1 *or* 0	subj rdf:type ?o	$\frac{c_{rdf:type}}{c_{rdf:type_{sub}}}$

Table D.1: Cardinality estimation of triple patterns

6.2 Cardinality Estimation of Joins

The join operation is performed on a common variable between two triple patterns. We consider three possible types of joins between two triple patterns based on the position of the common variable, namely: Subject-Subject (SS), Subject-Object (SO), and Object-Object (OO). If there is no common vari-

	Triple Pattern (TP)	DSC	DOC	E_{TP} Card	$E_{\bowtie}$ Card	$T_{\bowtie}$ Card
1:	?A *rdf:type* :FullProfessor	85,006	85,006	85,006		
2:	?A *:name* ?N	10,696,541	1,480	10,696,541	85,006	85,006
3:	?A *:teacherOf* ?C	359,795	1,079,580	1,079,580	8,579	255,148
4:	?C *:advisor* ?A	2,052,228	299,177	2,052,228	1,646	2,055,430
5:	?X *rdf:type* :GraduateCourse	539,467	539,467	539,467	822	1,027,909
6:	?X *rdf:type* :GraduateStudent	1,259,681	1,259,681	1,259,681	504	630,419
7:	?X *:degreeFrom* ?U	1,619,476	1,000	2,337,985	575	630,419
8:	?Y *:takesCourse* ?C	5,220,814	1,074,409	14,405,077	7,674	2,964,894
9:	?Y *rdf:type* :GraduateStudent	1,259,681	1,259,681	1,259,681	1,851	2,964,894
					$\sum$ =106,657	$\sum$ =10,614,119

(a) Join ordering using Global Statistics ($\mathcal{O}_{gs}$)

	Triple Pattern (TP)	DSC	DOC	E_{TP} Card	$E_{\bowtie}$ Card	$T_{\bowtie}$ Card
1:	?A *rdf:type* :FullProfessor	85,006	85,006	85,006		
2:	?A *:name* ?N	85,006	10	85,006	85,006	85,006
3:	?A *:teacherOf* ?C	85,006	255148	255,148	85,006	255,148
4:	?C *rdf:type* :GraduateCourse	539,467	539,467	539,467	255,148	255,148
5:	?X *:advisor* ?A	2,052,228	299,177	2,052,228	1,750,207	127,523
6:	?X *rdf:type* :GraduateStudent	1,259,681	1,259,681	1,259,681	1,074,297	1,027,909
7:	?X *:degreeFrom* ?U	1,259,681	1,000	1,259,681	659,416	630,419
8:	?Y *:takesCourse* ?C	5,220,814	1,074,409	5,220,814	8,841,082	2,964,894
9:	?Y *rdf:type* :GraduateStudent	1,259,681	1,259,681	1,259,681	2,133,181	2,964,894
					$\sum$ =14,883,343	$\sum$ =8,310,941

(b) Join ordering using Shapes Statistics ($\mathcal{O}_{ss}$)

Table D.2: This table shows the statistics (distinct subject count (DSC) and distinct object count (DOC)) of each triple pattern, the estimated cardinality of each triple pattern (E_{TP}), the estimated join cardinality ($E_{\bowtie\text{Card}}$) and the true join cardinality ($T_{\bowtie\text{Card}}$) for the ordered triple patterns of example query Q computed over LUBM dataset.

able between two triple patterns, the join will result in a Cartesian product. Inspired by related work [8], we estimate the SS, SO, and OO join cardinalities using the formulas stated in Equations D.1, D.2, and D.3. Note that DSC_i and DOC_i in the formulas represent the distinct subject and object count of triple pattern i respectively.

$$\widehat{card}\ (tp_i \ \bowtie_{SS} \ tp_j) \ = \ \frac{card_i \ \times card_j}{max\ (\ DSC_i\ ,\ DSC_j)} \tag{D.1}$$

$$\widehat{card}\ (tp_i \ \bowtie_{SO} \ tp_j) \ = \ \frac{card_i \ \times card_j}{max\ (\ DSC_i\ ,\ DOC_j)} \tag{D.2}$$

$$\widehat{card}\ (tp_i \ \bowtie_{OO} \ tp_j) \ = \ \frac{card_i \ \times card_j}{max\ (\ DOC_i\ ,\ DOC_j)} \tag{D.3}$$

6.3 Join Ordering

Given an RDF graph G, its shapes statistics graph (G_{sh}), and global statistics graph (G_{gs}), we propose an algorithm to compute the join ordering for an

Algorithm 3 Join Ordering

Input: Q, G, G_{sh} , G_{gs}
Output: Join order $\mathcal{O}$ of Q

$p \leftarrow [] \; ; r \leftarrow [] \;$; ▷ p: processed, r: remaining
$cost \leftarrow 0 \; ; card \leftarrow 0 \; ; queue \leftarrow queue.init();$
$tps \leftarrow getTPs(Q) \;$;
$tps_{\triangle} \leftarrow getCandidateShapes(Q, G, G_{sh}, G_{gs}) \;$;
$tps' \leftarrow computeCardinalities(tps_{\triangle}) \;$; ▷ Table D.1
$sort(asc, tps'.cardinality) \;$;
$p.add(tps'_0) \; ; r.addAll(tps' - tps'_0) \;$;
$cost = tps'_0.cardinality \;$;
$queue.add(tps'_0.index) \;$;
for $tp_i \in tps'$ **do** ▷ i > 0
  $index = tp_i.index \; ; cost_{local} = cost \;$;
  $queue' = queue \;$;
  while $!queue'.isEmpty$ **do**
    $tp_a = queue'.poll() \;$;
    for $tp_b \in r$ **do**
      $c = 0 \;$;
      if $\text{tp}_a \bowtie_T \text{tp}_b$ **then** ▷ $T \in \{SS, SO, OS, OO\}$
        $\text{c} = \bar{J}(tp_a, tp_b) \;$; ▷ $\bar{J} : T \times T \mapsto \mathbb{N}$ (Prob 2)
      else $c = cp(tp_a, tp_b) \;$; ▷ Cartesian Product
      if $c < cost_{local}$ **then**
        $cost_{local} = c \; ; index = tp_b.index \; ; card = c \;$;
      $cost\text{+=}cost_{local} \;$;
  $queue.add(index); p.add(tps'.get(index)) \;$;
  $r.remove(tps'.get(index)) \;$;
$\mathcal{O} \leftarrow queue.poll() \;$;

input query Q (Algorithm 3). In the first step, the triple patterns of Q are sorted in ascending order of their estimated cardinalities using only global statistics. The algorithm starts with the triple pattern having the least cardinality and then estimates its join cardinality with the rest of the triple patterns using the formulas from Section 6.2. The algorithm iterates over all the triple patterns and chooses a triple pattern with the least estimated join cardinality (size of intermediate result) given the triple already selected. This produces a first join ordering based on global statistics. In the second step, shapes statistics are taken into account, and both the estimated cardinalities and the join ordering proposed in the first step are revised using these shapes specific fine-grained statistics. The algorithm also computes the cost of each join ordering by adding the estimated join cardinalities in each iteration. Its complexity is cubic to the number of triple patterns in the query, i.e., $O(n^3)$.

Given our example query Q, and the cardinalities of its triple patterns T=$\{tp_1, tp_2, ..., tp_9\}$ estimated with both global and shape statistics, Tables D.2a and D.2b show the join ordering computed only using global statis-

tics ($\mathcal{O}_{gs}$) and via shapes statistics ($\mathcal{O}_{ss}$), respectively. There is a significant difference between the estimated and true join cardinalities and their final total cost. The estimated join cardinalities for $\mathcal{O}_{ss}$ are much closer to the true cardinalities of the query than the estimates for $\mathcal{O}_{gs}$ with two exceptions for tp_5 and tp_8 where shapes statistics largely overestimate their cardinalities due to skewed distribution of data.

7 Experimental Evaluation

We investigated the performance of query plans proposed using our algorithm (with global and shapes statistics) compared to the plans proposed by two state-of-the-art query engines (Apache Jena ARQ[3] and GraphDB[4]) as well as two state-of-the-art RDF cardinality estimation approaches (Characteristic Sets [19] and SumRDF [23]). All experiments are performed on a single machine with Ubuntu 18.04, having 16 cores and 256GB RAM.
Datasets: We used LUBM [10], WatDiv [2], and YAGO-4 [21] to study various query plans on different datasets and sizes (Table D.3). In particular, we used LUBM-500, two variants of WatDiv datasets (WATDIV-S (Small) with ~108.9 M triples and WATDIV-L (Large) with 1 billion triples), and for YAGO-4 we used the subset containing instances that have an English Wikipedia article.

	LUBM	WATDIV-S	WATDIV-L	YAGO-4
# of triples	91 M	108 M	1,092 M	210 M
# of distinct objects	12 M	9 M	92 M	126 M
# of distinct subjects	10 M	5 M	52 M	5 M
# of distinct RDF type triples	1 M	25 M	13 M	17 M
# of distinct RDF type objects	39	46	39	8,912

Table D.3: Size and characteristic of the datasets

Implementation: Nowadays, constraints languages are having widespread application to validate RDF graphs [21]. We assume the availability of SHACL shapes graph with the dataset and provide a *Shapes Annotator* to extend it with statistics of the graph. For cases where they are not present, the SHACLGEN[5] library is commonly used to generate shapes graphs and we also use it in our case (e.g., for YAGO-4). All shapes are then extended with the required statistics using our *Shapes Annotator* (implemented in Java). The SHACL shapes graph for LUBM, for instance, is 45 KB, and the size of extended shapes is 68 KB. The time required to extend the SHACL shapes depends on the number of its nodes and property shapes. The process of

[3] https://jena.apache.org/documentation/
[4] https://graphdb.ontotext.com
[5] https://pypi.org/project/shaclgen/

extending LUBM shapes graph took 16 minutes, WATDIV-S took 8 minutes, and for YAGO-4 (which consists of 8888 nodes and 80831 property shapes) it took 62 minutes. We implemented our join ordering algorithm in Java using Jena[3]. The source code is available on our website[6].

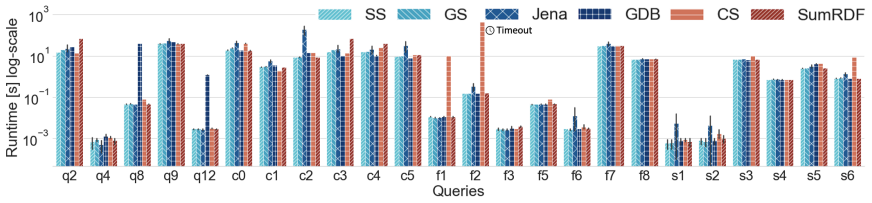
We loaded all three datasets and their relevant SHACL shapes graphs into Jena TDBs[3]. We used our join ordering algorithm to construct query plans using global and shapes statistics. For Jena, we used its ARQ query engine to obtain the query plans. For GraphDB, we loaded all datasets in GraphDB and used its *onto:explain* feature to obtain the query plans. For Characteristic Sets [19] approach, we generated characteristic sets of each dataset and used Extended Characteristic Sets [18] to optimize query plans for non-star type queries. Generating characteristic sets for large RDF graphs is computationally expensive. For instance, it took 6.2 hours to generate Characteristic Sets for LUBM, 1.2 hours for WATDIV-S, and 8.2 hours for YAGO-4.

For SumRDF [23], we generated the summaries of each dataset and adapted our join ordering algorithm to exploit their estimates. Similar to Characteristic Sets, the generated summaries require a few GBs of memory and their generation time depends on the size and heterogeneity of the dataset, e.g., it took 4.5 minutes to generate the summary for the LUBM, 14 minutes for WATDIV-S, and 4.3 hours for YAGO-4. We use the same size of LUBM and WatDiv datasets as used in SumRDF [23]. Hence, we used the same parameters to generate their summaries. It is suggested that a reasonable default size for the target SumRDF's summary should be in the order of tens of thousands [23]. Therefore, for YAGO-4, to generate the summary in a reasonable amount of time, we chose 100K as the target size of the summary.

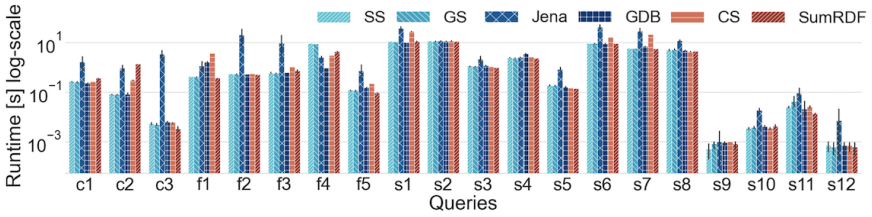
All query plans obtained using these approaches are executed 10x in Jena TDB and each query is interrupted after a timeout of 10 minutes. Since for some approaches the order in which triples are stated in the query matters we shuffle the triple patterns in the BGPs randomly in each iteration before proceeding with query optimization. As the query planning time is always less than 20 milliseconds for all approaches and queries, in the following we focus on analyzing the precision of the cardinality estimation and the resulting query performance.

Queries: We distinguish complex (C), snowflake (F), and star (S) queries. LUBM provides 14 default queries that have relatively simple structures. Therefore, we selected queries Q2, Q4, Q8, Q9, Q12 and then created a few additional queries for each category C, F, and S. The WatDiv benchmark includes 3 C, 7 S, and 5 F queries. For YAGO-4 there are no available standard queries or query logs available for benchmarking. Therefore, we have handcrafted 13 queries following the C, F, and S graph patterns from the WatDiv

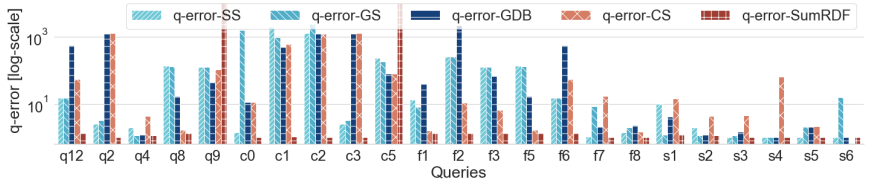
[6] https://relweb.cs.aau.dk/rdfshapes/



(a) Query runtime in LUBM

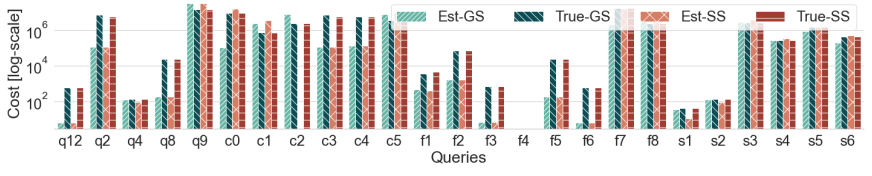


(b) Query runtime in YAGO-4

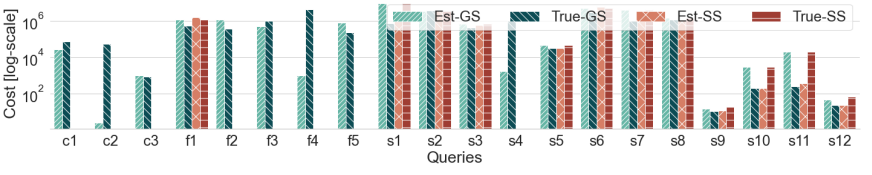


(c) q-error in LUBM

(d) q-error in YAGO-4



(e) Cost in LUBM



(f) Cost in YAGO-4

Fig. D.4: Query runtime, q-error, and cost analysis on LUBM and YAGO-4

Benchmark. These queries are available on our website[6].

Query Runtime: Due to space constraints, here we only report our findings on LUBM and YAGO-4, results on WatDiv datasets are discussed in the appendix of the extended version[6]. These experiments offer analogous insights to those obtained from the other datasets. Figure D.4a shows the query runtime analysis for query plans proposed using the SS approach *(plans constructed by our join ordering algorithm using shapes statistics)*, GS approach *(plans constructed by our join ordering algorithm using global statistics)*, Jena, GraphDB (GDB), Characteristic Sets (CS), and SumRDF on LUBM queries. The query runtime shows that: (i) the plans proposed by the SS approach are more efficient than those obtained with GS for queries having at least one type-defined triple pattern, (ii) the plans proposed by the GS approach are competitive in comparison to the plans of GDB, CS, and SumRDF, (iii) the CS approach is not well suited for large snowflake queries (e.g., F1, F2 (timeout), & F5), and (iv) the plans proposed by Jena are often suboptimal and non-deterministic (shown in the size of the error bars) as it is based on a heuristics-based query optimizer that takes into account the given order of triple patterns in the input query.

Similarly, Figure D.4b shows the query runtime for queries on YAGO-4. The query runtime for complex queries (C1, C2, C3) using SS and GS are competitive to the plans proposed by GDB, CS, and SumRDF. Snowflake queries provide interesting insights where each approach behaves differently for every single query. For instance, CS could not find the optimal query plan for queries F1, F3, F4, F5, and SS and GS could not find the most efficient query plan for query F4 due to underestimation of the join cardinalities. However, GDB and SumRDF found almost optimal query plans for all snowflake queries except F1 (GraphDB) and F4 (SumRDF). For star queries, almost all approaches identify plans with comparable good performances. Similar to LUBM, the plans proposed by Jena are rarely the most efficient.

In addition to query runtime, we also report the q-error, which is used to measure the precision of the final query result cardinality estimates [19]. It quantifies the ratio between the estimated ($\hat{c}$) and true result cardinality (c) and is computed as the ratio $max(max(1,c)/max(1,\hat{c}), max(1,\hat{c})/max(1,c))$. Ideally, the lower the value of the q-error, the better the estimates are. We analyze the q-error values for SS, GS, GDB, CS, and SumRDF. Figure D.4c shows the q-error analysis for LUBM queries. For SS, 15 queries have q-errors lower than 15, 8 queries have q-errors lowever than 250, and only 3 queries have q-errors greater than 250. For GS, 14 queries have q-errors lowever than 15, 8 queries have q-errors lower than 250, and only 4 queries have q-errors greater than 250. Overall, the q-errors for GS and SS are competitive to GDB and CS with few exceptions. However, overall the q-error is very low for SumRDF except queries Q9 and C5. Figure D.4d shows the q-error analysis for YAGO-4. For GS and SS, 14 queries have q-errors lower than 15, 2 queries

have q-errors lowever than 250, and only 4 queries have q-errors greater than 250. Similar to LUBM, the q-errors of GS and SS are competitive with GDB, CS, and SumRDF with few exceptions.

Finally, Figure D.4e and D.4f present the analysis between actual and true costs of query plans produced by SS and GS on the LUBM and YAGO-4 datasets. For LUBM, the cost estimated by SS is closer to the actual cost for Q4, Q9, C0, C1, C5, F7, F8, and all star queries. However, for YAGO-4, the cost estimated by SS is closer to the true cost for almost all queries except C2, F4, and S4.

Summary: Our results showed that, with only a few exceptions, the query plans proposed using SS and GS are competitive with the other tested approaches on both the synthetic and real data. Overall, the results revealed that our approach is efficient for all examined types of SPARQL queries while requiring only very little overhead to extend SHACL graphs with statistics, which is more efficient and feasible than generating extensive summaries or Characteristic Sets. On average, our approach finds the best query plans for 75% cases on both datasets. For the remaining cases, our approach proposes query plans having an overhead from 14% to 30% on average query runtime w.r.t. the best query plan. Our approach requires 2-4x less preprocessing time, this implies 2 to 6 hours less preprocessing time in our experiments, and 2 orders of magnitude less space.

8 Conclusion and Future Work

In this paper, we have presented an alternative approach to cardinality estimation for SPARQL query optimization. In particular, we have proposed novel light-weight statistics to capture the correlation in RDF graphs, a cardinality estimation approach, and a join ordering algorithm. We have performed extensive experiments on synthetic and real data to show our approach's effectiveness against two SPARQL query engines and two state-of-the-art RDF cardinality estimators. The results revealed that our approach is efficient in terms of both the preprocessing steps to generate statistics and the cardinality estimation to optimize query plans. Going forward, we plan to integrate our approach with one of the state-of-the-art query engines and enable the support of additional SPARQL query operators.

References

[1] A. Abbas, P. Genevès, C. Roisin, and N. Layaïda, "Selectivity estimation for sparql triple patterns with shape expressions," in *ICWE*, 2018, pp. 195–209.

[2] G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee, "Diversified stress testing of rdf data management." in *ISWC*, 2014, pp. 197–212.

[3] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, "Dbpedia-a crystallization point for the web of data," *Journal of web semantics*, vol. 7, no. 3, pp. 154–165, 2009.

[4] A. Bonifati, W. Martens, and T. Timm, "An analytical study of large sparql query logs," *The VLDB Journal*, vol. 29, no. 2, pp. 655–679, 2020.

[5] J. Broekstra, A. Kampman, and F. Harmelen, "Sesame: A generic architecture for storing and querying rdf and rdfs," in *ISWC*. Springer, 2002, pp. 54–68.

[6] W. Consortium, "Sparql 1.1 overview," https://www.w3.org/TR/sparql11-query/, 2013, accessed 23rd November, 2023.

[7] ——, "RDF 1.1," https://w3.org/RDF/, 2014, accessed 17th January, 2023.

[8] H. Garcia-Molina, J. D. Ullman, and J. Widom, *Database system implementation*. Prentice Hall Upper Saddle River, NJ, 2000, vol. 672.

[9] A. Gubichev and T. Neumann, "Exploiting the query structure for efficient join ordering in SPARQL queries," in *EDBT*, 2014, pp. 439–450.

[10] Y. Guo, Z. Pan, and J. Heflin, "Lubm: A benchmark for owl knowledge base systems," *Journal of Web Semantics*, vol. 3, pp. 158–182, 2005.

[11] S. Hagedorn, K. Hose, K. Sattler, and J. Umbrich, "Resource planning for SPARQL query execution on data sharing platforms," in *International Workshop (COLD) co-located with the 13th ISWC)*, vol. 1264, 2014.

[12] A. Hogan, "Shape constraints and expressions," in *The Web of Data*. Cham: Springer, 2020, pp. 449–513.

[13] H. Huang and C. Liu, "Estimating selectivity for joined RDF triple patterns," in *CIKM*. Glasgow, United Kingdom: ACM, 2011, pp. 1435–1444.

[14] H. Knublauch and D. Kontokostas, "Shapes constraint language (shacl)," *W3C Candidate Recommendation*, vol. 11, no. 8, 2017.

[15] V. Leis, B. Radke, A. Gubichev, A. Mirchev, P. A. Boncz, A. Kemper, and T. Neumann, "Query optimization through the looking glass, and what we found running the join order benchmark," *VLDB J.*, vol. 27, no. 5, pp. 643–668, 2018.

[16] B. McBride, "Jena: A semantic web toolkit," *IEEE*, vol. 6, no. 6, pp. 55–59, 2002.

[17] D. L. McGuinness, F. Van Harmelen *et al.*, "Owl web ontology language overview," *W3C recommendation*, 2004.

[18] M. Meimaris, G. Papastefanatos, N. Mamoulis, and I. Anagnostopoulos, "Extended characteristic sets: graph indexing for sparql query optimization," in *ICDE*. IEEE, 2017, pp. 497–508.

[19] T. Neumann and G. Moerkotte, "Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins," in *ICDE*. IEEE, 2011, pp. 984–994.

[20] Y. Park, S. Ko, S. S. Bhowmick, K. Kim, K. Hong, and W.-S. Han, "G-CARE: A Framework for Performance Benchmarking of Cardinality Estimation Techniques for Subgraph Matching," in *ACM SIGMOD*, 2020, pp. 1099–1114.

[21] T. Pellissier Tanon, G. Weikum, F. Suchanek *et al.*, "Yago 4: A reason-able knowledge base," in *ESWC*, 2020, pp. 583–596.

[22] E. Prud'hommeaux, J. E. L. Gayo, and H. Solbrig, "Shape expressions: an rdf validation and transformation language," in *ICSS*, 2014, pp. 32–40.

[23] G. Stefanoni, B. Motik, and E. V. Kostylev, "Estimating the cardinality of conjunctive queries over RDF data using graph summarisation," in *WWW*. ACM, 2018, pp. 1043–1052.

[24] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds, "Sparql basic graph pattern optimization using selectivity estimation," in *WWW*, 2008, pp. 595–604.

Paper E

Lossless Transformation of Knowledge Graphs to Property Graphs using Standardized Schemas

Kashif Rabbani, Matteo Lissandrini, Angela Bonifati,
Katja Hose

The paper is under review.