

UNIVERSITY OF OVIEDO



Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

DOCTORAL THESIS

Extraction of structured semantic knowledge through data mining over social media

Extracción de conocimiento semántico estructurado mediante minería de
medios sociales

Author/Autor:
Daniel Fernández-Álvarez

Supervisors/Directores:
Jose Emilio Labra Gayo
Daniel Gayo-Avello

*A thesis submitted in fulfillment of the requirements
for the degree of **Doctor of Philosophy***

Ph.D. Program/Programa de Doctorado: **Informática, RD 99/2011**

WESO Research Group
Department of Computer Science

May 23, 2022



RESUMEN DEL CONTENIDO DE TESIS DOCTORAL

1.- Título de la Tesis	
Español/Otro Idioma: Extracción de conocimiento semántico estructurado mediante minería de medios sociales	Inglés: Extraction of structured semantic knowledge through data mining over social media
2.- Autor	
Nombre: Daniel Fernández Álvarez	DNI/Pasaporte/NIE: 7
Programa de Doctorado: Informática	
Órgano responsable: Centro Internacional de Postgrado	

RESUMEN (en español)

Esta tesis se plantea inicialmente como una búsqueda de sistemas capaces de extraer RDF shapes a partir de minería de contenido textual expresados en medios sociales. Las shapes obtenidas con este sistema son una expresión formal y estructurada de conocimiento conceptual asociadas a un contexto de extracción concreto. Además de para tareas de descripción y validación de documentos, dicho conocimiento puede ser utilizado para realizar nuevas acciones que reviertan positivamente en la fuente minada, tales como clasificación automática de contenidos, sugerencia de contenidos o generación automática de plantillas y formularios.

La búsqueda de este sistema nos lleva a producir tres principales contribuciones: la arquitectura SEITMA, el sistema sheXer y el algoritmo ClassRank.

SEITMA es una arquitectura diseñada para extraer shapes a partir de lenguaje natural que combina dos procesos principales: la extracción de RDF a partir de lenguaje natural y la extracción de shapes a partir de minería de tripletas RDF. SEITMA se basa en la generalización de ejemplos: transforma textos con conocimiento a nivel de instancia en RDF shapes. Hemos implementado dos prototipos de SEITMA que se diferencian en su estrategia para extraer RDF de lenguaje natural. Hemos usado ambos prototipos para extraer shapes de resúmenes en páginas de Wikipedia. Las páginas escogidas se corresponden con instancias de clases importantes de DBpedia y cada shape representa conocimiento en Wikipedia asociado a estas clases. Si bien la naturaleza de las shapes producidas por cada uno de estos sistemas varía, ambos son capaces de obtener resultados prometedores, demostrando así la viabilidad de SEITMA.

sheXer es un sistema para la extracción automática de shapes a través del minado de grafos RDF. En el contexto de nuestra tesis, este sistema es usado por ambos prototipos de SEITMA. No obstante, la problemática resuelta por sheXer es de interés general para la comunidad de web semántica. Si bien existen varias alternativas para extraer RDF de lenguaje natural, sheXer es una propuesta competitiva, pues presenta un conjunto único de características entre los sistemas disponibles: 1) capacidad de generar contenido ShEx y SHACL, 2) capacidad para procesar grandes volúmenes de datos, 3) capacidad de generar shapes con rutas inversas y 4) capacidad para producir shapes que hacen referencia interna a otras shapes. sheXer usa una estrategia iterativa para procesar la información que permite manejar grandes grafos RDF. En este documento evaluamos el tiempo de ejecución y consumo de memoria de sheXer en experimentos en los que extraemos shapes de DBpedia, YAGO y Wikidata. Detectamos que el número de instancias relevantes para el proceso es el parámetro que más influye en el rendimiento, pero demostramos que una muestra representativa de instancias es suficiente para obtener resultados precisos.



ClassRank es un algoritmo que permite detectar las clases más importantes en un grafo RDF. ClassRank asocia a cada clase de un grafo una puntuación relacionada con la importancia acumulada de sus instancias. La importancia a nivel de instancia se calcula con PageRank. ClassRank es necesario para priorizar clases en distintas fases de los experimentos con prototipos SEITMA. No obstante, la detección de clases importantes en grafos RDF es un problema relevante más allá del contexto de nuestra tesis. Evaluamos la calidad de los rankings producidos por ClassRank midiendo su semejanza con rankings basados en frecuencia de uso de clases en logs de SPARQL. Las fuentes elegidas para la evaluación son Wikidata y DBpedia. Comparamos ClassRank con muchas otras alternativas existentes para medir importancia de clases y nuestra propuesta demuestra ser la que mejor se alinea con los rankings de frecuencia de uso.

El código fuente de ClassRank, sheXer y los prototipos SEITMA son públicos y todos los experimentos llevados a cabo en esta tesis pueden ser replicados.

RESUMEN (en Inglés)

The first motivation of this thesis is providing a system able to extract RDF shapes by mining text content expressed in social media.

Shapes obtained with such a system are a formal and structured expression of conceptual knowledge associated with a certain context. Besides being helpful for text validation and description, those shapes can be used to perform further actions that can be beneficial for the mined source, such as automatic content classification, content suggestion, or automatic generation of forms and templates.

While developing such a system, we produced three main contributions: the SEITMA architecture, the sheXer system, and the ClassRank algorithm.

SEITMA is an architecture designed for extracting shapes from natural language that combines two main processes: the extraction of RDF from natural language, and the extraction of shapes by mining RDF triples. SEITMA is based on example generalization: it uses examples of texts with instance-level knowledge to produce RDF shapes. We have implemented two prototypes of SEITMA that use different strategies for producing RDF from natural language. We used both prototypes to extract shapes from Wikipedia abstracts. Such abstracts are related to instances of important classes in DBpedia, and each shape generated contains Wikipedia knowledge associated with one of those classes. Our prototypes generate different types of shapes, but they both produce promising results that prove the feasibility of our proposal.

sheXer is a system for the automatic extraction of shapes by mining RDF graphs. In our thesis context, sheXer is used by both SEITMA prototypes. However, the specific problems tackled by sheXer are relevant for the Semantic Web community. Even if there are some other approaches for the automatic extraction of shapes from RDF content, sheXer is a competitive alternative, as it has a unique combination of features among existing alternatives: 1) generation of both ShEx and SHACL content, 2) capacity to process large graphs, 3) production of shapes with inverse paths, and 4) shape-interlinkage, i.e., production of shapes which can reference other shapes. sheXer uses an iterative strategy that avoids loading the whole target graph in main memory. We evaluated sheXer's performance w.r.t. execution time and memory usage. We detected that the number of instances relevant for the process is the parameter that affects the performance the most. However, we proved that sheXer produces precise results by using only representative instance samples.

ClassRank allows for detecting important classes in RDF graphs. ClassRank assigns each class an importance score w.r.t. the accumulated importance of its instances. The importance at instance-level is calculated using PageRank. We used ClassRank for class prioritization in several stages of the experiments with SEITMA prototypes. However, discovering the most



Universidad de Oviedo

important classes in RDF graphs is a problem relevant out of the context of our thesis. We have evaluated ClassRank by comparing the importance rankings that it produces with rankings based on class-usage in SPARQL logs. This evaluation was performed using Wikidata and DBpedia. We compared ClassRank with many other state-of-the-art approaches and demonstrated that our proposal is the one which produces the most similar results to the class-usage rankings.

The source code of ClassRank, sheXer, and the SEITMA prototypes has been publicly released, and every experiment described in this document can be replicated.

**SR. PRESIDENTE DE LA COMISIÓN ACADÉMICA DEL PROGRAMA DE DOCTORADO
EN INFORMÁTICA**

Abstract

The first motivation of this thesis is providing a system able to extract RDF shapes by mining text content expressed in social media.

Shapes obtained with such a system are a formal and structured expression of conceptual knowledge associated with a certain context. Besides being helpful for text validation and description, those shapes can be used to perform further actions that can be beneficial for the mined source, such as automatic content classification, content suggestion, or automatic generation of forms and templates.

While developing such a system, we produced three main contributions: the SEITMA architecture, the sheXer system, and the ClassRank algorithm.

SEITMA is an architecture designed for extracting shapes from natural language that combines two main processes: the extraction of RDF from natural language, and the extraction of shapes by mining RDF triples. SEITMA is based on example generalization: it uses examples of texts with instance-level knowledge to produce RDF shapes. We have implemented two prototypes of SEITMA that use different strategies for producing RDF from natural language. We used both prototypes to extract shapes from Wikipedia abstracts. Such abstracts are related to instances of important classes in DBpedia, and each shape generated contains Wikipedia knowledge associated with one of those classes. Our prototypes generate different types of shapes, but they both produce promising results that prove the feasibility of our proposal.

sheXer is a system for the automatic extraction of shapes by mining RDF graphs. In our thesis context, sheXer is used by both SEITMA prototypes. However, the specific problems tackled by sheXer are relevant for the Semantic Web community. Even if there are some other approaches for the automatic extraction of shapes from RDF content, sheXer is a competitive alternative, as it has a unique combination of features among existing alternatives: 1) generation of both ShEx and SHACL content, 2) capacity to process large graphs, 3) production of shapes with inverse paths, and 4) shape-interlinkage, i.e., production of shapes which can reference other shapes. sheXer uses an iterative strategy that avoids loading the whole target graph in main memory. We evaluated sheXer's performance w.r.t. execution time and memory usage. We detected that the number of instances relevant for the process is the parameter that affects the performance the most. However, we proved that sheXer produces precise results by using only representative instance samples.

ClassRank allows for detecting important classes in RDF graphs. ClassRank assigns each class an importance score w.r.t. the accumulated importance of its instances. The importance at instance-level is calculated using PageRank. We used ClassRank for class prioritization in several stages of the experiments with SEITMA prototypes. However, discovering the most important classes in RDF graphs is a problem relevant out of the context of our thesis. We have evaluated ClassRank by comparing the importance rankings that it produces with rankings based on class-usage in SPARQL logs. This evaluation was performed using Wikidata and DBpedia. We compared ClassRank with many other state-of-the-art approaches and demonstrated that our proposal is the one which produces the most similar results to the class-usage rankings.

The source code of ClassRank, sheXer, and the SEITMA prototypes has been publicly released, and every experiment described in this document can be replicated.

Resumen

Esta tesis se plantea inicialmente como una búsqueda de sistemas capaces de extraer RDF shapes a partir de minería de contenido textual expresados en medios sociales. Las shapes obtenidas con este sistema son una expresión formal y estructurada de conocimiento conceptual asociadas a un contexto de extracción concreto. Además de para tareas de descripción y validación de documentos, dicho conocimiento puede ser utilizado para realizar nuevas acciones que reviertan positivamente en la fuente minada, tales como clasificación automática de contenidos, sugerencia de contenidos o generación automática de plantillas y formularios.

La búsqueda de este sistema nos lleva a producir tres principales contribuciones: la arquitectura SEITMA, el sistema sheXer y el algoritmo ClassRank.

SEITMA es una arquitectura diseñada para extraer shapes a partir de lenguaje natural que combina dos procesos principales: la extracción de RDF a partir de lenguaje natural y la extracción de shapes a partir de minería de tripletas RDF. SEITMA se basa en la generalización de ejemplos: transforma textos con conocimiento a nivel de instancia en RDF shapes. Hemos implementado dos prototipos de SEITMA que se diferencian en su estrategia para extraer RDF de lenguaje natural. Hemos usado ambos prototipos para extraer shapes de resúmenes en páginas de Wikipedia. Las páginas escogidas se corresponden con instancias de clases importantes de DBpedia y cada shape representa conocimiento en Wikipedia asociado a estas clases. Si bien la naturaleza de las shapes producidas por cada uno de estos sistemas varía, ambos son capaces de obtener resultados prometedores, demostrando así la viabilidad de SEITMA.

sheXer es un sistema para la extracción automática de shapes a través del minado de grafos RDF. En el contexto de nuestra tesis, este sistema es usado por ambos prototipos de SEITMA. No obstante, la problemática resuelta por sheXer es de interés general para la comunidad de web semántica. Si bien existen varias alternativas para extraer RDF de lenguaje natural, sheXer es una propuesta competitiva, pues presenta un conjunto único de características entre los sistemas disponibles: 1) capacidad de generar contenido ShEx y SHACL, 2) capacidad para procesar grandes volúmenes de datos, 3) capacidad de generar shapes con rutas inversas y 4) capacidad para producir shapes que hacen referencia interna a otras shapes. sheXer usa una estrategia iterativa para procesar la información que permite manejar grandes grafos RDF. En este documento evaluamos el tiempo de ejecución y consumo de memoria de sheXer en experimentos en los que extraemos shapes de DBpedia, YAGO y Wikidata. Detectamos que el número de instancias relevantes para el proceso es el parámetro que más influye en el rendimiento, pero demostramos que una muestra representativa de instancias es suficiente para obtener resultados precisos.

ClassRank es un algoritmo que permite detectar las clases más importantes en un grafo RDF. ClassRank asocia a cada clase de un grafo una puntuación relacionada con la importancia acumulada de sus instancias. La importancia a nivel de instancia se calcula con PageRank. ClassRank es necesario para priorizar clases en distintas fases de los experimentos con prototipos SEITMA. No obstante, la detección de clases importantes en grafos RDF es un problema relevante más allá del contexto de nuestra tesis. Evaluamos la calidad de los rankings producidos por ClassRank midiendo su semejanza con rankings basados en frecuencia de uso de clases en logs de SPARQL. Las fuentes elegidas para la evaluación son Wikidata y DBpedia. Comparamos ClassRank con muchas otras alternativas existentes para medir

importancia de clases y nuestra propuesta demuestra ser la que mejor se alinea con los rankings de frecuencia de uso.

El código fuente de ClassRank, sheXer y los prototipos SEITMA son públicos y todos los experimentos llevados a cabo en esta tesis pueden ser replicados.

Acknowledgements

Doing a PhD is not easy. I think that the most difficult part of it is not performing the actual work, but being able to handle the huge amount of things that are not under your control. At least, when it is possible. I have met so many people that started this journey but, for different reasons, had to leave the boat before landing. And I have also met so many others that succeeded, but all of them, with no exception, had to fight their battles.

It took me seven years to finish this PhD. But, at least, I am lucky enough to be able to say that I made it. However, if I made it, it was thanks to the help and support of so many people that I would like to remember in this section of my work.

First of all, I want to thank Labra and Dani, my two PhD supervisors. Without their advice and technical orientation, especially during this intense last year, it would not have been possible to do this PhD.

I also want to mention all the people I knew during my research stay at the University of Leipzig. There, I learned a lot about research and Semantic Web. But, also, I lived three of the most exciting months of my life.

I want to thank my lab mates. Especially those that walked with me most of this way, Herminio, Óscar, and Cristian. But also those that were there before me, such as Javi, Manu, Jose, or Miguel, and the ones who come after, such as Willy and Pablo. And so many other people that I spent time with at the OOT-Lab. Their invaluable company and help have been essential in this process.

Without any doubt, the greatest support that I received came from my family. My parents, my sister, and, in their own way, my little nephews. Also all my family in Carcedo. Those of us who are here and those that are gone. I owe them absolutely EVERYTHING.

These years of thesis coincided with the advent of so many hardly avoidable responsibilities. Life goes on. But, despite all those changes, some friends keep being an anchor to reality, a table in the water, or a return home, as the times require. There are many people that I would like to mention here, but I must write at least the names of Carlos, Aida, and Maru.

And Alba. She was close to me just for the last part of this stage. But she became a fundamental pillar in my life. Someone who I hope to keep having by my side for the rest of my stages.

Finally, I would like to thank many students who I have already had the honor of teaching. In my case, the incredibly rewarding experiences lived in the classroom have been the beacon that I have needed in this path. One of the main reasons to keep thinking that doing a PhD is, indeed, worthy.

Agradecimientos

Hacer una tesis doctoral no es sencillo. Creo que la mayor dificultad de este proceso no reside en el esfuerzo neto necesario para llevar a cabo el trabajo, sino en ser capaz de reaccionar adecuadamente ante la ingente cantidad de cosas que no están bajo tu control. Si es que hay reacción posible. He podido a conocer a muchas personas tratando de seguir este camino que, por distintas razones, han tenido que bajarse del carro antes de llegar a destino. Y también he podido conocer a muchas otras que han conseguido llegar a meta y continuar con su carrera investigadora. Pero todas ellas, sin excepción, han tenido que librar sus batallas particulares.

A mi me ha llevado siete años llevar esta tesis a buen puerto, pero tengo la fortuna de poder decir que lo he logrado. Lo he logrado, eso sí, con la ayuda y el apoyo de muchas personas a las que les quiero dar las gracias en esta sección de mi trabajo.

En primer lugar, quiero agradecerlo a Labra y Dani, mis dos directores. Sin sus consejos y orientación técnica, especialmente durante este último año tan intenso, no habría sido posible finalizar esta tesis.

Quiero mencionar también a las personas que conocí durante mi estancia en la Universidad de Leipzig. Allí no solo aprendí mucho sobre investigación y Web Semántica, sino que además pude vivir tres de los meses más emocionantes de mi vida.

Quiero dar las gracias a mis compañeros de laboratorio. Muy especialmente a aquellos con los que he recorrido casi todo este camino, Herminio, Óscar y Cristian. Pero también a los que estaban antes que yo, como Javi, Manu, Jose o Miguel, y a los que vienen, como Willy y Pablo. Y muchas otras personas con las que he coincidido en el OOT-Lab. Su inestimable compañía y ayuda han sido esenciales en este proceso.

En cuanto a apoyo, el mayor de todos ha venido, sin lugar a dudas, de mi familia. Mis padres, mi hermana y, a su manera, mis sobrinos pequeños. Toda mi familia en Carcedo. Los que estamos y los que nos han faltado. A ellos les debo absolutamente TODO.

Estos años de tesis han coincidido con un avance en etapas vitales que conlleva una inevitable escalada de responsabilidades. A pesar de tantos cambios, hay amigos que permanecen como un ancla a la realidad, una tabla en el agua o una vuelta a casa, según la situación lo requiera. Hay muchas personas que querría mencionar aquí, pero no puedo dejar de acordarme de Carlos, Goti y Maru.

Y Alba. Alba solo me ha acompañado de cerca al final de esta etapa. Pero se ha convertido en un pilar fundamental de mi vida. Uno de cuya compañía espero poder seguir presumiendo en las etapas que me queden.

Por último, quiero dar las gracias a muchos estudiantes a los que ya he tenido el placer de dar clase. En mi caso, las increíblemente gratificantes experiencias dentro del aula han sido el faro que he necesitado en este camino. Una de las razones con más peso para seguir creyendo que terminar una tesis doctoral, de hecho, merece la pena.

Contents

Abstract	iii
Resumen	v
Acknowledgements	vii
Agradecimientos	ix
1 Introduction	1
1.1 Motivation	1
1.2 Research Questions	7
1.3 Summary of contributions	9
1.4 Document structure	10
2 Theoretical Framework	11
2.1 Semantic Web and Linked Data	11
2.1.1 Linked Open Data	16
2.2 RDF	18
2.2.1 Different syntaxes	21
2.2.2 Extending the expressiveness of basic triples	24
2.2.3 ‘Schemaless’ character	25
2.3 Knowledge Graphs	26
2.4 SPARQL	28
2.5 Ontology	30
2.5.1 Assertion Box and Terminological Box	33
2.6 RDF Shapes	33
2.6.1 Shape Expressions	35
2.6.2 Shape Constraint Language	39
2.6.3 Brief comparison between SHACL and ShEx	44
2.7 Social Media	46
2.7.1 Wikipedia	48
2.8 Natural Language Processing	50
2.8.1 NLP on social media	55
2.9 Data mining	56
2.9.1 Data mining in social media	59
3 Importance metrics in RDF graphs	61
3.1 Introduction	61
3.2 Metrics	64
3.2.1 Importance metrics applied over schema structures	65
3.2.1.1 Degree	65
3.2.1.2 Betweenness	65
3.2.1.3 Bridging Centrality	65

3.2.1.4	Closeness and Harmonic Centrality	65
3.2.1.5	Radiality	66
3.2.2	Importance metrics applied over the whole graph structure	66
3.2.2.1	Instance counting	66
3.2.2.2	PageRank	67
3.2.2.3	HITS	67
3.2.2.4	ClassRank	67
3.2.3	Adapted importance metrics	72
3.3	Experiments	72
3.3.1	Methodology	73
3.3.1.1	Reference rankings: human-generated traffic vs machine-generated traffic	73
3.3.1.2	How to compare the rankings	74
3.3.2	Sources	75
3.3.2.1	DBpedia	75
3.3.2.2	Wikidata	78
3.3.3	Results	82
3.4	Discussion	87
3.4.1	Best performing techniques	87
3.4.2	General performing of techniques in different sources	89
3.4.3	EH vs HH results	90
3.4.4	OTT, AAT and adapted metrics	90
3.4.5	Configuration of class-pointers	90
3.4.6	Computational cost vs performance	91
3.5	Related work	93
3.5.1	Scoring entities or classes in graphs	93
3.5.2	Alternative centrality measures based in PageRank	95
3.6	Conclusions	97
3.6.1	Future Work	98
4	Mining triples to extract shapes	99
4.1	Introduction	99
4.2	System description	103
4.2.1	Graph Iterator	105
4.2.2	Instance tracker	105
4.2.3	Feature Tracker	106
4.2.4	Shape Adapter	107
4.2.4.1	Extending cardinalities	109
4.2.4.2	Sorting triple constraints	109
4.2.4.3	Removing empty shapes	110
4.2.5	Shape Serializer	110
4.2.6	Computational complexity analysis	110
4.3	Experiments	111
4.3.1	Limiting the number of instances used	113
4.3.2	Limiting the number of target shapes	113
4.3.3	Limiting the amount of triples	114
4.3.4	Convergence w.r.t. number of instances used	115
4.3.4.1	Discussion on execution times	117
4.4	sheXer working modes	119
4.4.1	Input types	119
4.4.1.1	Formats	119

4.4.1.2	Input types	120
4.4.1.3	Independent class-instance file	121
4.4.1.4	Working with endpoints	121
4.4.2	Target entities and shapes	123
4.4.2.1	Finding target entities in endpoints	124
4.4.2.2	Compatibility between modes	125
4.4.3	Management of instantiation property	126
4.4.4	Namespaces management	126
4.4.5	Configuring output	127
4.4.6	Acceptance threshold	127
4.4.7	Inverse paths	128
4.4.8	All-compliant mode	129
4.4.9	Removing empty shapes	129
4.4.10	Cardinality prioritization	130
4.4.11	Adaptation to Wikidata model	130
4.4.11.1	Readable results	130
4.4.11.2	Handling qualifiers	130
4.5	Related work	134
4.6	Conclusions	135
4.6.1	Future work	137
5	Mining shapes from social media	139
5.1	Introduction	139
5.2	Architecture description	141
5.3	Prototypes	143
5.3.1	Prototype 1: SEITMA-L	143
5.3.1.1	LAREWA	143
5.3.1.2	SEITMA-L implementation	146
5.3.2	Prototype 2: SEITMA-F	148
5.3.2.1	FRED	148
5.3.2.2	SEITMA-F implementation	151
5.4	Experiments	153
5.4.1	Experiments with SEITMA-L	155
5.4.1.1	Inputs and settings	155
5.4.1.2	Outputs	157
5.4.1.3	Discussion about SEITMA-L features	165
5.4.2	Experiments with SEITMA-F	168
5.4.2.1	Inputs and settings	168
5.4.2.2	Outputs	169
5.4.2.3	Discussion about SEITMA-F features	179
5.5	Discussion on use cases	181
5.5.1	Use cases for Wikipedia	182
5.5.1.1	Automatic generation of templates	182
5.5.1.2	Content suggestion	183
5.5.1.3	Vandalism detection	183
5.5.2	Use cases out of Wikipedia	184
5.5.2.1	Text validation and class summarization in formal environments	184
5.5.2.2	Automatic types or tags	184
5.5.2.3	Application in text streams	185
5.6	Related work	186

5.7	Conclusions	188
5.7.1	Future work	189
6	Conclusions	191
6.1	Conclusions (English version)	191
6.2	Conclusiones (Versión en castellano)	193
A	Prefix definitions	197
	Bibliography	199

List of Figures

1.1	Web layers	2
1.2	Interpretations of a piece of data contained in a document by humans and machines.	3
1.3	Relations between <i>Oviedo, Spain</i> , and <i>city</i> .	3
1.4	Nodes <i>Jimbo</i> and <i>Sarah</i>	5
1.5	Example of a shape <code><Person></code> written in ShEx.	5
2.1	Content served by a web browser when asking for the URI <code>http://dbpedia.org/resource/Tim_Berners-Lee</code>	13
2.2	Content served by the command <code>curl</code> using content negotiation with the URI <code>http://dbpedia.org/resource/Tim_Berners-Lee</code>	13
2.3	Several URIs equivalent to <code>http://dbpedia.org/resource/Tim_Berners-Lee</code>	15
2.4	Several types of the URI <code>http://dbpedia.org/resource/Tim_Berners-Lee</code>	15
2.5	LOD Cloud, May 2007. Image credits to <code>lod-cloud.net</code>	17
2.6	LOD Cloud, March 2009. Image credits to <code>lod-cloud.net</code>	18
2.7	LOD Cloud, May 2021. Image credits to <code>lod-cloud.net</code>	19
2.8	Example of RDF/XML syntax	21
2.9	Basic example of turtle syntax	22
2.10	Reification example: expressing the notion “ <i>A is 20 km far from B</i> ”	25
2.11	Basic example of SPARQL	29
2.12	ShEx example: a person should have a <code>foaf:name</code> .	36
2.13	Turtle example: a person with a <code>foaf:name</code> .	36
2.14	Shape map example	37
2.15	ShExC example content	37
2.16	SHACL example in turtle syntax	41
2.17	Partial example vectors for the words <i>driver</i> , <i>pilot</i> , and <i>cactus</i> .	54
3.1	Comparison of techniques against EH log in DBpedia.	83
3.2	Comparison of techniques against HH log in DBpedia.	84
3.3	Comparison of techniques against EH log in Wikidata.	85
3.4	Comparison of techniques against HH log in Wikidata.	86
3.5	Performance comparison of ClassRank against any other non-ClassRank metric at every source and prefix depth.	87
3.6	Performance comparison of Adapted ClassRank against any other non-ClassRank metric at every source and prefix depth.	88
3.7	Performance comparison of Adapted ClassRank against any other non-ClassRank metric at every source and prefix depth.	89
4.1	Example shape of Country in ShExC	99
4.2	Content about some countries in turtle.	100
4.3	sheXer base architecture.	104

4.4	Example of constraints that could get positive votes from a certain triple.	106
4.5	Fragment of the DBpedia's shape :TelevisionShow in ShExC	113
4.6	Performance of sheXer with different amounts of instances used.	114
4.7	sheXer's with different number of target shapes.	115
4.8	Performance of sheXer with different dataset sizes.	116
4.9	Shape convergence using different amounts of instances per shape.	117
4.10	Python code - Basic example of sheXer usage	119
4.11	Python code - Change input format	120
4.12	Python code - Standalone file for instance-class relations	122
4.13	Subgraphs reachable according to different endpoint configurations	123
4.14	Python code - Consuming and endpoint with depth=1 + extra classes	124
4.15	Python code - Shape maps and all classes mode together	126
4.16	Python code - Feeding sheXer with namespace-prefix pairs	127
4.17	Python code - Generation of SHACL content	127
4.18	Python code - Setting an acceptance threshold	128
4.19	Python code - Enabling inverse paths	128
4.20	Python code - Disabling all-compliant mode	129
4.21	Python code - Keeping empty shapes	129
4.22	Python code - Disabling exact cardinality	130
4.23	('Q30 - United states', 'P6 - head of government', 'Q6279 - Joe Biden') using both direct and non-direct Wikidata properties	131
4.24	Python code - Extracting Wikidata shapes with direct properties	132
4.25	Python code - Extracting shapes for Wikidata qualifiers	133
5.1	SEITMA core	141
5.2	SEITMA's C1. General solution for an ML approach	142
5.3	SEITMA-L: implementation based on a language-agnostic ML approach	146
5.4	DRS representation of "Larry likes coffee. If someone likes it, he drinks it."	149
5.5	SEITMA-F: implementation based on FRED	151
5.6	:Band. A shortened example of a shape produced by SEITMA-L	158
5.7	Part of Metallica's Wikipedia page.	161
5.8	Cardinality, node constraint, and constraint direction w.r.t. focus node of TCs in SEITMA-L results.	162
5.9	Cardinality and node constraint of comments in SEITMA-L results.	163
5.10	Distribution of trustworthy scores among TCs in SEITMA-L results.	164
5.11	Distribution of trustworthy scores among comments in SEITMA-L re- sults.	165
5.12	:Band. A shortened example of a shape produced by SEITMA-F	170
5.13	Cardinality, node constraint, and constraint direction w.r.t. focus node of TCs in SEITMA-F target shapes.	174
5.14	Cardinality and node constraint of comments in SEITMA-F target shapes.	175
5.15	Distribution of trustworthy scores among TCs in SEITMA-F target shapes.	175
5.16	Distribution of trustworthy scores among comments in SEITMA-F tar- get shapes.	176
5.17	Cardinality, node constraint, and constraint direction w.r.t. focus node of TCs in shapes produced by SEITMA-F.	177
5.18	Cardinality and node constraint of comments in shapes produced by SEITMA-F.	177

5.19	Distribution of trustworthy scores among TCs in shapes produced by SEITMA-F.	177
5.20	Distribution of trustworthy scores among comments in shapes produced by SEITMA-F.	178
5.21	Section of the shape <i>:Artists</i> produced by SEITMA-F	180
5.22	Partial example of FRED's output.	180
5.23	Possible mapping for the content shown in Figure 5.22.	181

List of Tables

3.1	Statistics about the DBpedia SPARQL logs used	76
3.2	Top20 elements for HH and EH entries in DBpedia	79
3.3	Statistics about the Wikidata SPARQL logs used	80
3.4	Properties with a class ratio ≥ 0.99	81
3.5	Top20 elements for HH and EH entries in Wikidata	82
3.6	Computational cost of the techniques evaluated	91
3.7	Top 20 of ClassRank and Instance Counting	92
4.1	Basic information about the YAGO, Wikidata and DBpedia computa- tions.	112
4.2	Shape convergence with different instance limits per class.	118
5.1	Summary data about automatic classifiers in SEITMA-L experiments. .	157
5.2	Statistics about shapes in SEITMA-L results	162
5.3	Statistics about shapes in SEITMA-F results	172

List of Abbreviations

AAT	Also Assertion Techniques
AI	Artificial Intelligence
API	Application Programming Interface
A-BOX	Assertion Box
BNode	Blank Node
CIP	Constraint-Instance Performance
CNN	Convolutional Neural Network
COCA	Corpus of Contemporary American English
CRF	Conditional Random Fields
CSV	Comma Separated Values
CWA	Closed World Assumption
DHCP	Dynamic Host Configuration Protocol
DC	Dublin Core
DL	Description Logics
DoS	Denial-of-Service
DRS	Discourse Representation Structure
DUL	DOLCE+DnS Ultra Lite
EH	Every Host
EL	Entity Linking
ER	Entity-Relationship
FAIR	Findability Accessibility Interoperability Reusability
FL	Figurative Language
FOAF	Friend of a Friend
FT	Feature Tracker
GA	Genetic Algorithms
GB	Giga Bytes
GI	Graph Iterator
HH	Human Hosts
HMM	Hidden Markov Models
HTTP	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IC	Instance Counting
ICV	Integrity Constraint Validation
IOP	Inverse Open Path
IP	Internet Protocol
IR	Information Retrieval
IT	Instance Tracker
IRI	Internationalized Resource Identifier
I/O	Input/Output
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation for Linked Data
KB	Knowledge Base

KBP	Knowledge Base Population
KG	Knowledge Graph
LAREWA	Language-Agnostic Relation Extraction from Wikipedia Abstracts
LD	Linked Data
LLM	Large Language Models
LM	Language Model
LOD	Linked Oamed Data
MB	Mega Bytes
MH	Machine Hosts
ML	Machine Learning
MO	Music Ontology
NE	Named Entity
NEC	Named Entity Classification
NEL	Named Entity Linking
NER	Named Entity Recognition
NIF	NLP Interchange Format
NN	Neural Network
NLP	Natural Language Processing
NT	N-triples
N3	Notation 3
ODP	Ontology Design Patterns
OPRL	OpenPath Rule Learner
OTT	Only Terminological Techniques
OWA	Open World Assumption
OWL	Web Ontology Language
POS	Part of Speech
PPR	Personalized PageRank
PR	Property-Role
RAM	Random-Access Memory
RBO	Rank-Biased Overalp
RDF	Resource Description Framework
RDFS	RDF Vocabulary Description Language
RE	Relation Extraction
REST	Representational State Transfer
RML	RDF Mapping Language
RNN	Recurrent Neural Network
RQ	Reseachr Question
SA	Shape Adapter
SEITMA	Shape Extraction from Instance Text Mining Architecture
SHACL	Shapes Constraint Language
ShEx	Shape Expressions
SIP	Sshape-Instance Performance
SNS	Social Network Site
SPARQL	SPARQL Protocol and RDF Query Language
SPIN	SPARQL Inferencing Notation
SRL	Semantic Role Labeling
SS	Shape Serializer
SVM	Support Vector Machines
SWD	Semantic Web Documents
TC	Triple Constraint
T-BOX	Terminological Box

UNA	Unique Name Assumption
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URN	Uniform Resource Name
URL	Uniform Resource Locator
WESO	Web Semantics Oviedo
WSD	Word Sense Disambiguation
W3C	World Wide Web Consortium
XML	eXtended Markup Language
XSLT	eXtensible Stylesheet Language Transformations

Chapter 1

Introduction

1.1 Motivation

Since the proposal of *the World Wide Web* in 1989 [1] and its advent in the early 1990s, many milestones had a great impact on the way in which the Web was used. For example, the appearance in the late 1990s of Google's search engine, built on top of the PageRank algorithm [2], was a remarkable event, as the web content become much easier to find for the users. However, it was not until 2004 that the term *Web 2.0* was coined.

This "2.0" label, which suggests a determinant evolution of the notion of Web, was first used in a conference brainstorming session between O'Reilly and MediaLive International [3]. The key event that motivated the label Web 2.0 was the proliferation of web sites such as blogs, forums, and social network sites. Those platforms have a common feature: they are built around user-generated content. The Web before those systems consisted mostly of static pages whose content was written by their owners or maintainers. In contrast, Web 2.0 platforms allow for creating web content via user actions and interactions. This type of systems are known as *social media*.

Many authors have provided definitions, sometimes contradictory, of what exactly social media is [4–7]. However, nowadays, the great importance of social media is widely acknowledged, as "*a large proportion of contemporary communication takes place through it*"[4].

There is a huge volume of data being posted on social media. For example, a report on Facebook statistics at the time of this writing (January 2022)¹ shows that that platform has 2.912 billion monthly active users, representing 36.8% of the global population. Also, 1.929 billion of them have daily activity on the platform. A similar analysis on Twitter (February 2022)² indicates that 500 million tweets are posted every day.

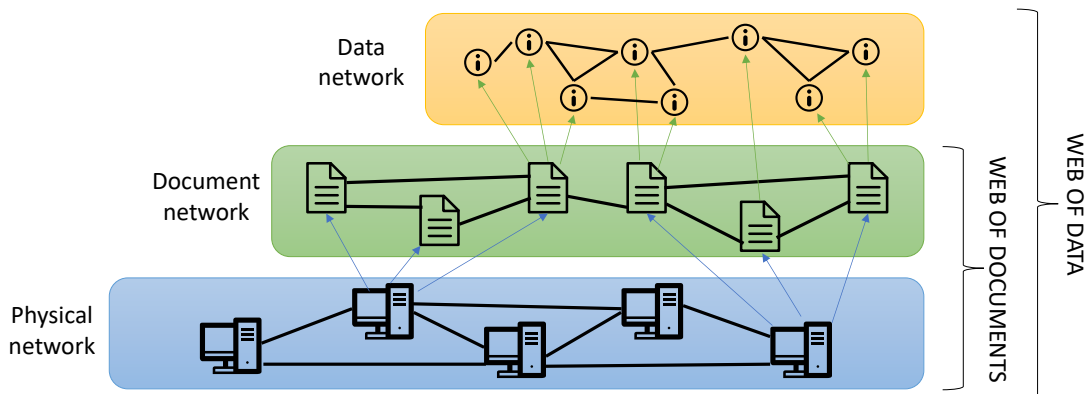
Needless to say, the amount of data generated in a single day would be impossible to read by a human in a lifetime. Besides, it is also difficult to obtain actionable knowledge from this data using automatic processes. Most content is published in unstructured formats, usually as plain text pieces of natural language. Converting those pieces of natural language into actionable information so it can be used by automatic agents implies dealing with issues such as:

- Computing large corpora usually requires special hardware and software solutions.
- Not all the information is relevant.

¹<https://datareportal.com/essential-facebook-stats> Accessed in 2022/05/03.

²<https://www.omnicoreagency.com/twitter-statistics/> Accessed in 2022/05/03.

FIGURE 1.1: Web layers



- Some pieces of information can be contradictory.
- Some pieces of information can be wrong.

The goal of *Natural Language Processing (NLP)* techniques is to process human languages so they can be “understood” by machines. Additionally, *data mining* techniques aim to extract relevant notions from large volumes of data. Frequently, being able to obtain valuable knowledge from a large text corpus requires approaches that make use of both NLP and data mining techniques.

Social media content has been mined for a wide variety of purposes [8–18]. The results of those mining processes can be conveyed in several ways: different types of structured data, charts, text summaries, lists information snippets, etc. Deciding an adequate representation format for some information is especially relevant when one wants to publicly share the data obtained, so it can be consumed by other users. This decision is even more important when one wants this data to be reusable by automatic processes too. For such a case, Resource Description Framework (RDF) can be used as a *de facto* standard to represent and share structured information.

RDF is a technology for data representation associated to the so-called *Semantic Web*. Essentially, this term refers to an evolution from a web of documents to a web of data. In figure 1.1, we show a graphical representation of this evolution. The Web is *hosted* by an actual physical network of interconnected hardware devices. Each device can host different documents (or, in general, resources). One of the fundamental ideas of the Web since its creation is to identify each resource with a unique Universal Resource Locator (URL). Using HyperText Transfer Protocol (HTTP) [19] and some associated technologies, web browsers can retrieve the appropriate content for a certain URL. Web pages link to each other using their respective URLs, so the Web can be seen as a directed graph of resources.

Such documents can be retrieved by machines, but they cannot be understood by them. This idea is illustrated in Figure 1.2. One of the core proposals of the Semantic Web consists in assigning unique identifiers to actual entities and concepts, such as the country *Spain*, the city *Oviedo*, or the abstract idea of *city*. Relations between those entities and concepts can be expressed using named links between them. For example, one could link the nodes *Spain* and *Oviedo* with a relation such as *located at*, or the nodes *Oviedo* and *city* with a relation such as *is a*, as it is shown in Figure 1.3.

FIGURE 1.2: Interpretations of a piece of data contained in a document by humans and machines.

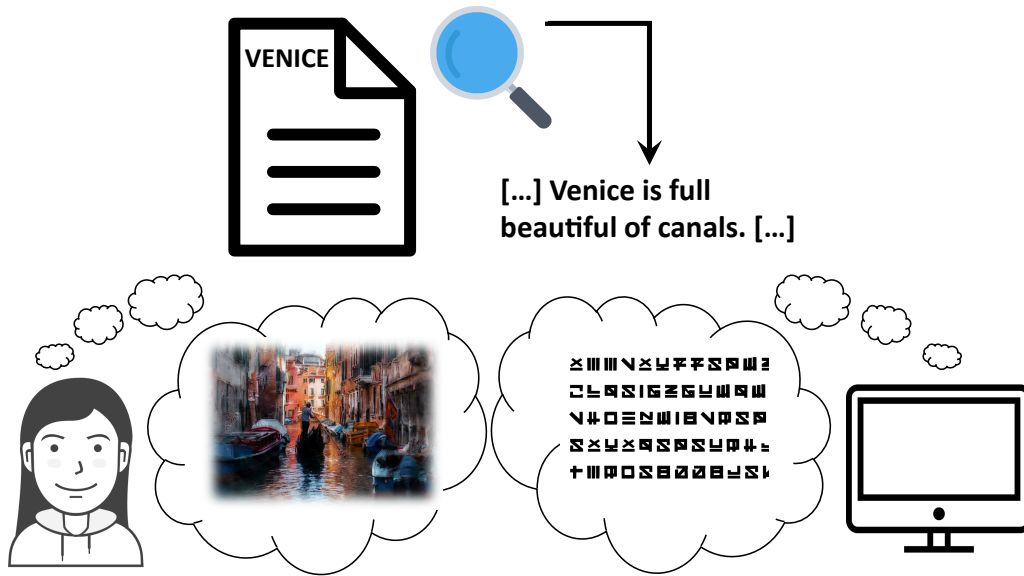
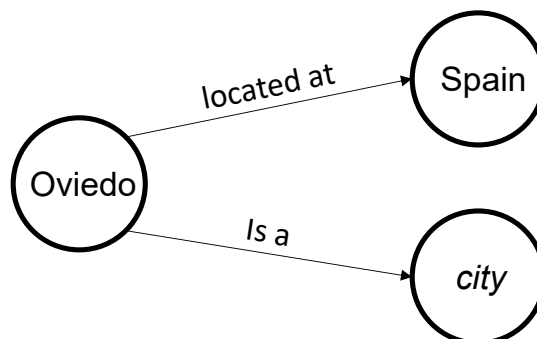


FIGURE 1.3: Relations between *Oviedo*, *Spain*, and *city*.



The properties used to link elements are also associated to unique identifiers. Such entities and properties can be declared within actual RDF documents or exposed in SPARQL endpoints³.

With this, we evolve from a directed graph of documents to a concept of Web consisting of a directed graph of entities linked with named relations. While documents cannot be understood by machines, web sites built on top of Semantic Web ideas can generate views that can be both readable by humans and consumed by automatic agents. The potential of this idea is that it allows for using the Web as a kind of huge database where every piece of data could be interconnected.

RDF is the standard language to express entity graphs in web environments. A key feature of RDF that makes it suitable for being a *lingua franca* for sharing many different types of data is its *schemaless* profile. In RDF graphs, unlike in traditional databases, the entities are not required to adapt a strict schema, i.e., they can be potentially linked with any other piece of data.

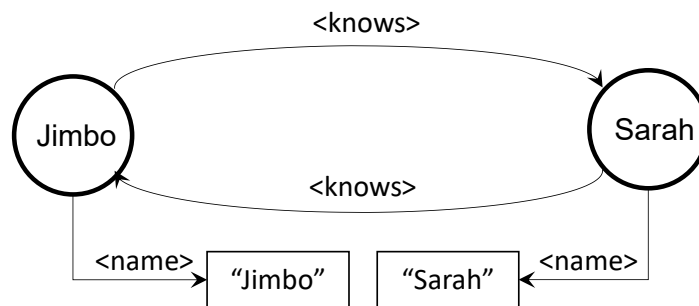
However, it is important to remark that, even if RDF graphs are not necessarily enforced to have strict schemata, they *should* have schemata. How the information is connected must be predictable. For example, let us say that there is a property p which is used in a graph to express a relation *is a* between two nodes a and A . Then, one may be able to assume that any other *is a* relation between a different pair of entities b and B will be expressed using p . Also, if there is a widely accepted property p' used in many other RDF graphs to express the relation *is a*, then one should use p' instead of p or, at least, express that $p = p'$ using RDF. For example, let us say that we have an RDF graph describing cities. Then, one should be able to assume things such as:

- Nodes which are cities will be linked with an *is a* relation with the node *city*.
- Nodes that are cities are expected to be linked with other nodes using some set of known relations, such as *population*, *surface*, or *nearest city*.
- If a node c_1 which is a city is linked with another node c_2 using the relation *nearest city*, then c_2 should be a city too.

Without those kinds of agreements and assumptions, the Semantic Web would be just a bunch of interconnected data whose inconsistent structure would not allow for obtaining valuable knowledge from it. RDF is called *schemaless* because it works with flexible schemata, not because it lacks of them. RDF data can be easily adapted by adding new relations to an existing graph. Also, different RDF sources can be easily integrated by merely establishing links between entities from separated graphs. This simple action allows for merging the data and the latent schemata of both graphs.

Shape languages, such as Shapes Constraint Language (SHACL) [20] and Shape Expressions (ShEx) [21], have emerged during the last years to fulfill needs w.r.t. schemata in RDF content. These languages are based on the concept of *shape*. Essentially, shapes are formal expressions of expected features for a certain type of node in the context of an RDF graph. For example, one could use a shape to indicate that those nodes that represent a *Person* must be associated with exactly one string using the relation named *name*. The shape could also indicate that *Person* nodes may be connected with other nodes using the relation *knows*, as long as those nodes are *Person* nodes too. In Figure 1.4, we show a couple of nodes *Jimbo* and *Sarah* that meet such criteria. Then, in Figure 1.5, we show a shape $\langle Person \rangle$ that can be used

³SPARQL and SPARQL endpoint will be described in section 2.4.

FIGURE 1.4: Nodes *Jimbo* and *Sarah*FIGURE 1.5: Example of a shape *<Person>* written in ShEx.

```

1
2 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
3 BASE <http://www.example.org/>
4
5 <Person> {
6   <name> xsd:string ;
7   <knows> @<Person>
8 }
9

```

to describe the mentioned features and to validate that *Jimbo* and *Sarah* have their expected topology. This example is written using ShEx⁴.

Currently, there are two main use cases in which shape languages are being used: validation and description of RDF sources. On the one hand, for a certain graph, one could use shape languages to check whether the expectations described for the nodes are actually met. For that task, it is necessary to use a *shape validator*, i.e., a software which interprets a shape language, receives an input graph, and is able to discover and report constraint violations in this graph. Such reports help to discover unexpected features in RDF graphs that may need to be addressed. For example, constraint violations affecting a node *n* could be attended by modifying the links of *n* or by adapting applications which are supposed to consume content related to *n*.

On the other hand, shape languages can have an eminent documentation purpose. RDF graphs are often queried using *SPARQL Protocol and RDF Query Language* (SPARQL, recursive acronym). Essentially, this query language is based on definitions of graph patterns where some of the elements (nodes, properties) are unknown variables. When new users of a given SPARQL endpoint write their first queries, it usually takes some time until they realize which is the actual structure of the explored data. One needs to know this structure, i.e., which type of nodes are supposed to connect to other nodes using which properties, to write effective queries. Instead of trying exploratory SPARQL queries, or reading textual documentation, shape languages can be used as excellent communication tools to describe the expected neighborhood for the different types of entities available at any endpoint.

Even though the current usage of shape languages is confined to RDF environments, we think shapes can be beneficial in other contexts apart from RDF validation and description. In the same way that RDF can be used to formalize and share content which was originally unstructured, shape languages can be used as a formal

⁴In section 2.6, we will describe how to use and interpret shape languages.

expression of conceptual knowledge in sources of unstructured information. Moreover, our hypothesis is that shapes obtained by mining sources of unstructured content can be used as a base to perform further actions with automatic processes. We think that this is especially interesting when applied to large sources where too many individual agents are producing content, such as in social media. Some examples of potential uses of shapes in such contexts are:

- **Generating forms or templates.** Let us suppose that the expected minimal information provided for one or more classes in Wikipedia is known and expressed with shapes. For example, let us suppose that there is a shape associated to the concept *athlete* indicating that pages of athletes should include information such as specialty, citizenship, championships, and world ranking. That shape could be used to produce text templates or forms allowing for the creation of a first version of the Wikipedia page of a new athlete.
- **Automatic classification.** Let us say there is a discussion in a forum about a public person, and many statements about that person are written by different users. Let us also say there are several shapes describing different occupations, such as *politician*, *drummer*, or *journalist*. The knowledge associated with that person could be gathered and compared with the existing shapes. With that, we would be able to 1) know the main occupation of that person, and 2) know how to represent in RDF their associated information in a way that is homogeneous with that from other people with the same occupation.
- **Content suggestion.** Let us suppose there is a consistent post structure in a blogging platform, which is known and expressed using a shape. Then, that shape could be used to suggest adding some parts which a new post is missing. Moreover, if that post is about an entity that can be associated with an identifier used in a given RDF graph, those suggestions may not be merely structural, but they could also provide actual values associated to the post's topic.

However, the automatic generation of shapes out of social media content, or, in general, any kind of natural language, has not received much attention from the research community yet. To the best of our knowledge, only a system to produce direct mappings between shapes and text content has been proposed [22], and the text pieces that can be processed are actual descriptions of shapes expressed in natural language. That system is expected to be fed with sentences such as “*Every user has exactly one username*” or “*Each student has at least one subject enrolled*”. Such approach allows for quickly writing shapes using natural language rather than formal syntaxes. However, that type of sentences may be hard to be found in user-generated content whose primary intent is not describing a shape.

Our premise is that, in social media, partial descriptions of actual entities are more frequent than descriptions of abstract contents. For example, one could hardly expect to find a Twitter user explaining how *soccer players* should be in terms like “*Soccer players are born in a country. They can play for a single soccer team at a time. They are usually specialized in a position*”. In contrast, messages such as “*The Argentinian forward Lionel Messi has just signed with Paris Saint-Germain!*”, which conveys exactly the information related to country, team, and position in an actual entity, are more likely to be found. If we are able to mine enough examples of features associated to soccer players, then we would be able to extract a shape containing the most frequent features associated to those athletes.

Our thesis begun with this seed idea. We wanted to develop systems able to automatically extract RDF shapes from social media content by generalizing features

observed at instance level. As already stated, such shapes could improve the information or user experience in the platforms where the input data was mined. They could also be used as a standard formal tool to describe and share sets of expected features and structures. As so, they can help to make maintenance decisions w.r.t. data quality.

1.2 Research Questions

The work developed in this thesis is motivated by the following Research Question (RQ):

- **RQ1:** How can we automatically extract shapes from social media content?

As already stated in section 1.1, the existing methods to transform text into RDF shapes are not well suited the type of texts usually found in social media. The possible courses of action to tackle the problem formulated in **RQ1** can be roughly classified in two types of strategies. On the one hand, as suggested in [22], we can try to create direct mappings between natural language and RDF shapes. On the other hand, we could first convert natural language into some intermediary representation, and then transform that representation into RDF shapes. Our initial intuition on this matter is that using the later approach is more promising, especially if we use RDF as intermediary format. This is justified as follows:

- A *machine reader* is a tool that transform natural language into some sort of structured knowledge. Machine reading is a very active research field [23]. Particularly, several solutions have been proposed to transform text into content expressed using Semantic Web technologies [24]. If we can use or adapt existing machine readers to map natural language to RDF, then we could simplify the problem of **RQ1**.
- The intermediate RDF content used to extract shapes would not be an expendable product, but reused as a structured representation of the target text.
- RDF shapes are mainly tools to validate and describe RDF content. Thus, the shapes obtained can increase their value if they can be used for validation purposes too. In this case, the validation of a text t with a shape s could be indirectly produced by validating an RDF representation of t with s .

We decided to split the problem of extracting shapes from natural language in two different subproblems: 1) getting RDF content from natural language, and 2) getting RDF shapes from RDF content. As already stated, there are several existing approaches to tackle the problems related to the first task [24]. Unfortunately, at the moment of starting the work on this thesis, little had been done to extract shapes from RDF graphs.

It is important to remark that writing and maintaining RDF shapes is a costly process. In spite of the many – and above described – advantages of RDF, most of the Web’s knowledge is still represented in unstructured or semi-structured formats. This is because producing and maintaining RDF data is not a trivial task. The issues related to manually create shapes are similar to the issues of manually create RDF triples: “*there is simply too much information available on the Web – information that is constantly changing – for it to be feasible to apply manual annotation to even a significant subset of what might be of relevance*” [24].

The same problems affect the manual production of shapes: “when a KG⁵ contains hundreds of classes, each having many attributes, manually specifying (post-hoc) the necessary shapes becomes a tedious and unmanageable task” [25].

In this scenario, to provide a proper answer for **RQ1**, a new RQ appears:

- **RQ2:** How can we produce shapes by mining RDF triples?

RQ2 was born as a needed solution to solve problems related to **RQ1**. Still, we realized that the potential applications of eventual approaches proposed to answer **RQ2** could be useful on their own and not just as a mean to solve **RQ1**, as the automatic generation of shapes from existing RDF content is relevant for the Semantic Web community.

One of our initial intuitions was that using available knowledge on existing RDF sources could be helpful to solve problems related with **RQ1**. Even if the volume of data published using Semantic Web standards keeps growing⁶, the amount of data available as RDF cannot be compared with the amount of content produced in social media as free text.

We thought that the decision on what knowledge domains consider to test our proposals should be driven by the type of data available in the RDF sources used to support the extraction. That is, we should choose working with “important topics” for the RDF sources. In this context, the adjective *important* conveys different notions, such as abundant, complete, well-connected, correct, etc. Regarding the word *topic*, it could be interpreted in several ways when applied to RDF graphs. For example, each of the entities identified in a graph could be treated as an individual topic. However, we think that such notion of topic would have been too fine-grained. Instead, we thought that equating the notions of *topic* and *class* could be much more convenient in our scenario. A class (e.g., *country*) can be used as representative of the knowledge accumulated by its instances (e.g., *Spain, India, Japan*). Also, both **RQ1** and **RQ2** are oriented to produce shapes, and shapes are usually associated to classes. Therefore, shapes associated with important classes of an RDF graph could also be important for the consumers or maintainers of such graph.

Many existing approaches based on *graph centrality* have been proposed to measure node importance in different types of networks [28]. However, none of the existing approaches was specifically designed to measure the importance of classes in RDF graphs, so a third RQ emerges from this situation:

- **RQ3:** How can we identify the most important classes of an RDF graph?

To answer this question, we needed to evaluate the existing proposals according to our needs. In case none of the approaches meet those proposals, then we should propose new ways to measure class importance in RDF graphs.

As with **RQ2**, **RQ3** is oriented to solve a problem related to the context of this thesis. However, any work performed to answer **RQ3** could also be useful in studies of class importance unrelated to **RQ1**.

So, in short, this thesis aims to provide answers for three main research questions. **RQ1** was the question initially motivating our work. However, even if **RQ2** and **RQ3** are originally framed within **RQ1**, these two questions involve interesting

⁵Acronym of *Knowledge Graph*. In this context, it refers to RDF graphs.

⁶Truth is there are some existing public general-purpose sources containing millions of triples about many different topics [26, 27]. In the following portal, one can check the evolution of sites publishing data with Semantic Web standards and a public license in different years <https://lod-cloud.net/> Accessed in 2022/05/03.

challenges appearing in many other application contexts. Along this document, we provide thorough studies to answer **RQ2** and **RQ3**, and we evaluate our proposals w.r.t. this questions in scenarios which are not dependent on **RQ1**.

1.3 Summary of contributions

In this section, we summarize the main contributions produced while answering the RQs posed above. We present these contributions in the same order in which they will be further developed along this document.

With regards to **RQ3**:

- We have designed a new algorithm, *ClassRank*, to measure class importance in RDF graphs. ClassRank assigns importance scores to classes w.r.t. the accumulated PageRank scores of their instances. This approach is able to capture the importance of classes with few instances when those instances are important for the graph structure as a whole.
- We have performed an experiment comparing ClassRank and several state-of-the-art approaches for measuring class importance. This experiment checks the similarity between the ranking produced by each technique with several model rankings of importance. As suggested in [28], those rankings are built by measuring class usage in SPARQL logs. In our experiments, ClassRank performed better than any other technique.
- We provide an implementation of ClassRank for the Python programming language. This implementation has been publicly released and is free to use.

With regards to **RQ2**:

- We have designed a new system, *sheXer*, to extract shapes from RDF content, whose core ideas are:
 - Generalizing the most frequent features from a group of instances to produce a shape as specific as possible satisfying all of them.
 - Using a voting system to detect frequent features. Each feature observed for each instance is used to cast positive votes for potential constraints in the shapes associated to that instance.
 - Designed to work with large RDF graphs, such as DBpedia or Wikidata. It uses an iterative approach that avoids allocating in main memory any content that is not relevant for the process.
 - Highly configurable. It accepts different kinds of inputs, it provides several options to customize the extraction process, and it can produce both ShEx and SHACL content.
- We have performed a study to evaluate the performance of *sheXer* w.r.t. memory usage and execution time using a number of large and well known RDF datasets. This study shows that *sheXer*'s memory usage has a linear relation with the number of relevant instances and target shapes, but it is independent of the target graph's size. This allows for processing big real-world datasets using inexpensive hardware.
- Our Python implementation of *sheXer* is also public and free to use.

With regards to **RQ1**:

- We have proposed an architecture, *SEITMA*, to extract shapes from pieces of natural language based on example generalization. *SEITMA* describes how to combine methods to extract triples from natural language with methods to extract shapes from RDF graphs.
- We have implemented two different Python prototypes following the *SEITMA* architecture. Both use *sheXer* as shape extractor, but they rely on different solutions to extract triples from natural language.
- We have performed some experiments with these two prototypes. We extracted shapes associated to the most important classes in DBpedia (according to ClassRank) using Wikipedia abstracts. Both prototypes were able to extract shapes from those abstracts.
- The source code of both prototypes has been publicly released and is free to use.
- We have described and discussed several use cases where shapes produced with *SEITMA* can be helpful. Also, we have discussed the features that a *SEITMA* implementation should have to be successfully used in such scenarios.

1.4 Document structure

In chapter 2, we provide a theoretical framework including definitions of concepts that are used along this work. We also provide a literature review on those concepts.

The next three chapters further develop the three RQs posed in section 1.2. Each chapter provides its own literature review focused on the chapter's specific RQ. They also include an introduction and partial conclusions related to that RQ.

Due to the dependencies among the RQs, the contents are developed in the reverse order in which the questions were stated. That is, chapter 3 is focused on class importance metrics which are related to **RQ3**: we describe our ClassRank algorithm and evaluate our proposal and other state-of-the-art techniques. In chapter 4, we address problems related to **RQ2**: we describe and evaluate *sheXer*, and we also review and discuss other approaches to perform shape extraction from RDF triples. Then, in chapter 5, we develop the contents related to question **RQ1**: we describe and evaluate *SEITMA*, its implemented prototypes, the experimental results obtained, and several use cases for the architecture.

Finally, in chapter 6, we provide a compilation of the general conclusions of our work.

Chapter 2

Theoretical Framework

2.1 Semantic Web and Linked Data

The *Semantic Web*, also known as *Web of Data*, proposes a new form of Web content in which one can refer and link specific data rather than mere documents. Such data should be represented in a machine-readable format, so diverse applications (not browsers) can re-use it, sometimes in unplanned ways [29].

The term Linked Data refers to a set of good practices to publish data. These practices or principles were initially proposed by Tim Berners-Lee in order to support the implementation of the Semantic Web, and consist of the following ones:

1. Use Universal Resource Identifiers (URIs) [30] to name things.
2. Use HyperText Transfer Protocol (HTTP) [19] URIs, so people can easily look up those things.
3. When someone looks up a URI, provide useful information about it, using standard technologies (RDF, SPARQL).
4. Include links to other URIs, so that they can discover more things.

The terms *Semantic Web* and *Linked Data* are not equivalent but are strongly connected. While Semantic Web refers to a global vision and philosophy of how the Web of interconnected data should be, Linked Data describes the basic principles to articulate this vision.

The principle number '1' proposes to use URIs as a standard way to identify elements. The usage of URIs makes sense in the Web context, as the Web prior to Semantic Web widely used Uniform Resource Locator (URLs) to link documents. URLs are a specialization of URIs that explicitly indicate how to retrieve the document referred (they are necessarily bounded to an Internet protocol so the document can be located and retrieved). By definition, URIs do not necessarily need to specify a way to retrieve a document, but they can be used as mere identifiers associated with a certain entity or concept. The specialization of URIs aiming to provide an identifier but not a way to retrieve the data associated to the entity that they represent are called Uniform Resource Names (URNs).

The principle number '2' indicates that HTTP URIs should be used, i.e., URIs indicating that the HTTP protocol is required to retrieve the data that they identify. Essentially, Linked Data is about “*using the Web to create typed links between data from different sources*” [31]. To achieve this data integration, there should be a way to retrieve the data. The choice of HTTP, again, makes sense as it is the Web's standard de facto to exchange information. URIs and HTTP are the two basic bricks used to build the whole Linked Data ecosystem. URIs that also allow for retrieving some

content using browsers are called *dereferenceable*. HTTP allows for using dereferenceable URIs to identify elements. That is, dereferenceable URIs are, at a time, a name to universally identify an entity and a way to retrieve data associated to that entity.

The principle number ‘3’ describes how the information about those things identified with URIs should be provided. The W3C standard Resource Description Framework (RDF) is critical for such a task. In traditional Web, HyperText Markup Language (HTML) provides a way to format and link documents with hyperlinks. In the Web of Data, RDF provides a way to describe entities and concepts with a generic, graph-based data model. The minimal information unit in RDF is the triple. Triples are associations of three elements, named *subject*, *predicate*, and *object*. Each triple expresses a relation between two elements (subject and object) by means of a named relation (predicate). For instance, a notion such as “*Berners-Lee is a person*” can be represented with a triple such as:

- **Subject:** http://dbpedia.org/resource/Tim_Berners-Lee.
- **Predicate:** <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>.
- **Object:** <http://dbpedia.org/ontology/Person>.

RDF content can be served with dereferenceable URIs. Note that a certain HTTP URI could serve different content depending on the headers of the performed HTTP request. For example, if the subject URI of the previous example is written in a common web browser, an HTML document will be obtained. This HTML document is a representation of the RDF information which lets the browser display it in a human-friendly way, as it is shown in Figure 2.1¹. However, with content negotiation², the same server asked for the same URI can retrieve actual RDF content, as shown in Figure 2.2³.

Some other standards are used on top of RDF to describe and access the data. On the one hand, ontologies allow to define reusable vocabularies (classes and properties) which let describe RDF nodes and even use logical inference on RDF graphs. The base languages to write ontologies are RDF Vocabulary Description Language (RDFS) [32], and Ontology Web Language (OWL) [33]. These languages are expressed using RDF too. Anyone can define a new vocabulary to model a certain knowledge domain, and those new vocabularies can be connected to elements of other existent vocabularies. The definition of reusable vocabularies follows the same principles of Linked Data.

On the other hand, we have SPARQL. SPARQL is the standard query language for RDF. SPARQL defines a framework to describe graph patterns in which some of the nodes (or relations) are variables. A query engine supporting SPARQL should be able to parse those patterns and retrieve the pieces of data that match the variables. It is highly recommended to expose RDF in SPARQL endpoints, i.e., endpoints that can solve SPARQL queries. SPARQL endpoints allow both to effectively access the content of an RDF graph and to integrate and use together the information exposed

¹Petition performed with Mozilla Firefox. Accessed in 2022/05/03

²Enabling content negotiation to serve the contents associated to an URI in different formats is not mandatory. However, it is a common practice of portals holding Linked Data. It is up to each portal whether it implements content negotiation, and, if so, which are the offered formats.

³The command used to retrieve this content is `curl -L -H 'Accept: application/n-triples' http://dbpedia.org/resource/Tim_Berners-Lee`. The header indicated in this command asks for n-triples content, which is one of the several syntaxes in which RDF can be serialized. Accessed in 2022/05/03

FIGURE 2.1: Content served by a web browser when asking for the URI http://dbpedia.org/resource/Tim_Berners-Lee



The screenshot shows the DBpedia page for Tim Berners-Lee. At the top, there is a navigation bar with the DBpedia logo and options like 'Browse using' and 'Formats'. Below that, the title 'About: Tim Berners-Lee' is displayed. A short biography follows, mentioning his birth in London in 1955 and his role as the inventor of the World Wide Web. To the right of the text is a portrait of Tim Berners-Lee. Below the biography is a table with two columns: 'Property' and 'Value'. The first row shows the 'dbpedia:abstract' property with a detailed description of his work and contributions to the World Wide Web.

Property	Value
dbpedia:abstract	<ul style="list-style-type: none"> Timothy "Tim" John Berners-Lee (Londres, Reino Unido, 8 de junio de 1955), es un científico de la computación británico, conocido por ser el padre de la World Wide Web. Estableció la primera comunicación entre un cliente y un servidor usando el protocolo HTTPS en diciembre de 1990. En octubre de 1994 fundó el Consorcio de la World Wide Web (W3C) con sede en el MIT, para supervisar y estandarizar el desarrollo de las tecnologías sobre las que se fundamenta la Web y que permiten el funcionamiento de Internet. Ante la necesidad de distribuir e intercambiar información acerca de sus investigaciones de una manera más efectiva, Berners-Lee desarrolló las ideas fundamentales que estructuran la web. Él y su grupo crearon lo que por sus siglas en inglés se denomina Lenguaje HTML (HyperText Markup Language) o lenguaje de etiquetas de hipertexto, el protocolo HTTP (HyperText Transfer Protocol) y el sistema de localización de objetos en la web URL (Uniform Resource Locator). Es posible encontrar muchas de las ideas plasmadas por Berners-Lee en el proyecto Xanadú (que propuso Ted Nelson) y el memex (de Vannevar Bush). (es) Sir Timothy John Berners-Lee OM KBE FRS FREng FRSA FBCS (born 8 June 1955), also known as TimBL, is an English computer scientist best known as the inventor of the World Wide Web. He is a Professorial Fellow of Computer Science at the University of Oxford and a professor at the

FIGURE 2.2: Content served by the command `curl` using content negotiation with the URI http://dbpedia.org/resource/Tim_Berners-Lee

```
<http://dbpedia.org/resource/Tim_Berners-Lee> <http://www.w3.org/2002/07/owl#sameAs>
<http://eu.dbpedia.org/resource/Tim_Berners-Lee> .
<http://dbpedia.org/resource/Tim_Berners-Lee> <http://www.w3.org/2002/07/owl#sameAs>
<http://it.dbpedia.org/resource/Tim_Berners-Lee> .
<http://dbpedia.org/resource/Tim_Berners-Lee> <http://dbpedia.org/ontology/wikiPageExternalLink>
<http://news.bbc.co.uk/2/hi/technology/4132752.stm> .
<http://dbpedia.org/resource/Tim_Berners-Lee> <http://xmlns.com/foaf/0.1/depiction>
<http://commons.wikimedia.org/wiki/Special:FilePath/Berners-Lee_announcing_W3F.jpg> .
<http://dbpedia.org/resource/Tim_Berners-Lee> <http://www.w3.org/2000/01/rdf-schema#label>
"Tim Berners-Lee"@pl .
<http://dbpedia.org/resource/Tim_Berners-Lee> <http://www.w3.org/2000/01/rdf-schema#comment>
"Timothy \u201ETim\u201C John Berners-Lee (* 8. \u010Derвна 1955 Lond\u00FDn) je
anglick\u00FD informatik, tv\u016Frce World Wide Webu a \u0159editel konsorcia W3C, kte
r\u00E9 dohl\u00ED na pokra\u010Duj\u00EDc\u00ED v\u00FDvoj webu."@cs .
<http://dbpedia.org/resource/Tim_Berners-Lee> <http://dbpedia.org/ontology/wikiPageWikiLink>
<http://dbpedia.org/resource/Department_of_Computer_Science,_University_of_Oxford> .
<http://dbpedia.org/resource/Tim_Berners-Lee> <http://dbpedia.org/ontology/wikiPageWikiLink>
<http://dbpedia.org/resource/Contract_for_the_Web> .
<http://dbpedia.org/resource/Tim_Berners-Lee> <http://dbpedia.org/property/wikiPageUsesTemplate>
<http://dbpedia.org/resource/Template:Youtube> .
<http://dbpedia.org/resource/Tim_Berners-Lee> <http://dbpedia.org/ontology/wikiPageWikiLink>
<http://dbpedia.org/resource/Category:Fellows_of_the_American_Academy_of_Arts_and_Sciences> .
```


in different graphs. RDF, RDFS, OWL, and SPARQL are described in detail in sections 2.2, 2.4, and 2.5 of this chapter.

Finally, the principle number ‘4’ indicates that the content provided should include links to URIs in other sources. This is a key idea to produce Linked Data instead of just RDF content. Representing and exposing data following the principles 1 to 3 (URIs, RDF and SPARQL) allows for exposing data in a machine-readable way. However, if the content does not include links to URIs of external sources, this framework is not much better than a traditional database with a flexible data model, as it does not enable to implement an actual interconnected Web of Data.

The RDF content associated to Berners-Lee in DBpedia contains several examples of how to link sources. On the one hand, one can say that a certain node is considered equivalent to some other node of an external source, i.e., both nodes depict the same entity or concept. The standard way to declare that an URI is equivalent to some other URI is using the property named `owl:sameAs`⁴, which is defined in OWL. In Figure 2.3, we show several examples of URIs declared to be equivalent to Berners-Lee’s URI in DBpedia⁵. As one can see, those URIs point to entities in diverse sources, such as specific language chapters of DBpedia, Freebase [34], Cyc[35] or Viaf [36]. The projects mentioned are all LD sources. With this, one may be able to easily merge the content about Berners-Lee available in different projects and use it in a single SPARQL query that combines the information of such sources.

On the other hand, one can link a URI of an outer graph with an internal URI by means of any named relation (the URI of the relation itself may belong to an external ontology). In Figure 2.4, we show several examples of URIs declared to be the type⁶ of Berners-Lee’s node in DBpedia⁷. As one can see, there are references to class URIs in ontologies such as FOAF [37], YAGO [38], Schema⁸, or Wikidata [26]. One may want to query information relative to instances of a class such as `schema:Person`. The information about these instances, and even the triple that declare that an instance’s type is `schema:Person`, can be exposed in different SPARQL endpoints and yet be used in a single query.

In order to execute a query which integrates content from different endpoints, a SPARQL engine supporting and enabling *federation* [39] should be used. There are other mechanisms to integrate different sources, including local batch processing of RDF dumps. However, all these mechanisms are based on the existence of links between URIs.

In summary, in Berners-Lee’s words “*The Semantic Web isn’t just about putting data on the Web. It is about making links, so that a person or machine can explore the Web of data. With linked data, when you have some of it, you can find other, related, data*” [40]. The Linked Data principles are thought to enable the implementation of this global vision.

⁴`owl:sameAs` is equivalent to the URI <http://www.w3.org/2002/07/owl#sameAs>. Some RDF syntaxes allow for defining prefixes that can be mapped to namespaces, which makes URIs shorter and thus more human-readable. The prefix `owl` is usually paired with the namespace <http://www.w3.org/2002/07/owl#>. This topic will be developed in section 2.2.

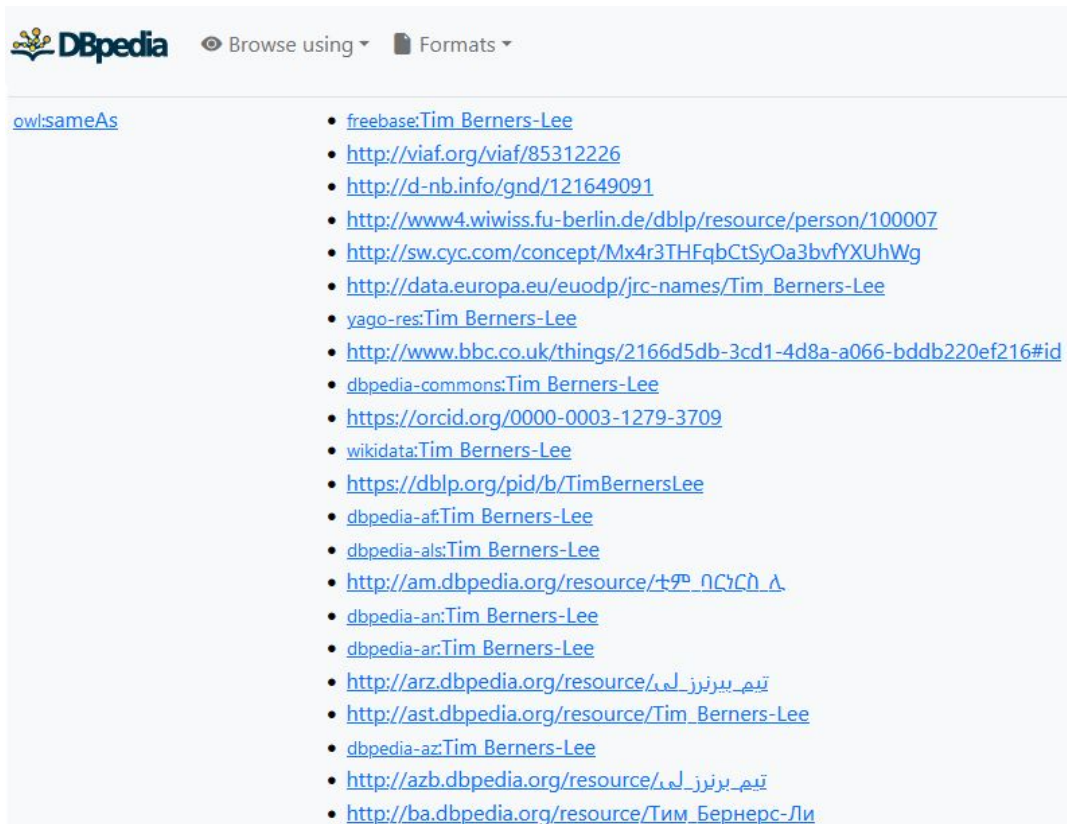
⁵https://dbpedia.org/page/Tim_Berners-Lee Accessed in 2022/05/03

⁶The standard way to declare the type of an URI in RDF is using the property <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>.

⁷https://dbpedia.org/page/Tim_Berners-Lee Accessed in 2022/05/03.

⁸<https://schema.org/docs/about.html> Accessed in 2022/05/03.

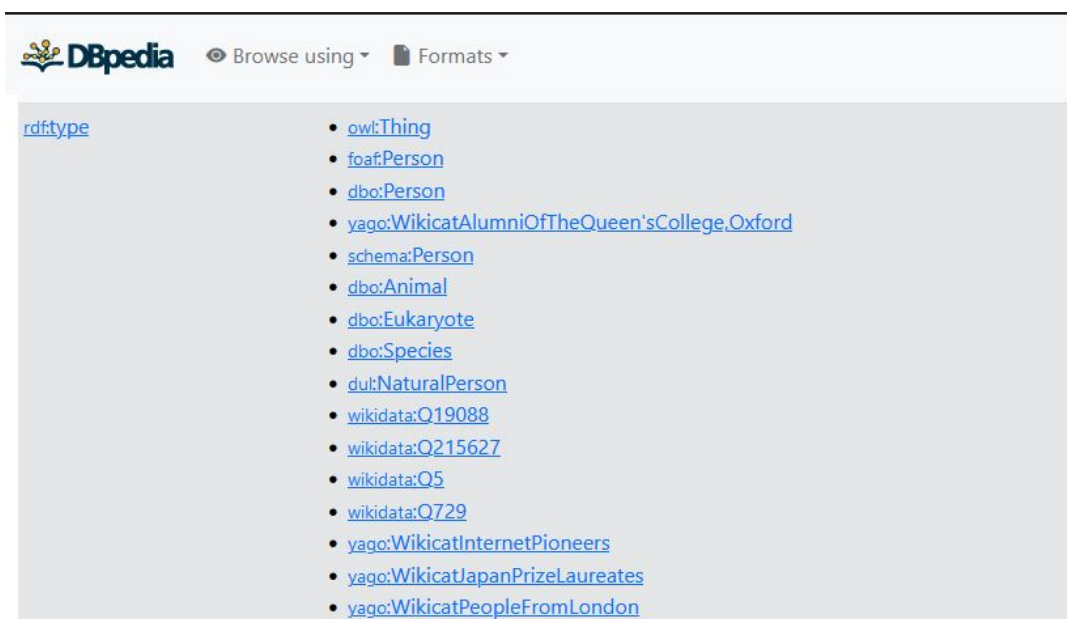
FIGURE 2.3: Several URIs equivalent to `http://dbpedia.org/resource/Tim_Berners-Lee`



The screenshot shows the DBpedia interface for the URI `http://dbpedia.org/resource/Tim_Berners-Lee`. The page displays the DBpedia logo and navigation options. Under the `owl:sameAs` property, a list of 20 equivalent URIs is provided, including:

- `freebase:Tim Berners-Lee`
- `http://viaf.org/viaf/85312226`
- `http://d-nb.info/gnd/121649091`
- `http://www4.wiwiss.fu-berlin.de/dblp/resource/person/100007`
- `http://sw.cyc.com/concept/Mx4r3THFqbCtSyOa3bvfYXUhWg`
- `http://data.europa.eu/euodp/jrc-names/Tim_Berners-Lee`
- `yago-res:Tim Berners-Lee`
- `http://www.bbc.co.uk/things/2166d5db-3cd1-4d8a-a066-bddb220ef216#id`
- `dbpedia-commons:Tim Berners-Lee`
- `https://orcid.org/0000-0003-1279-3709`
- `wikidata:Tim Berners-Lee`
- `https://dblp.org/pid/b/TimBernersLee`
- `dbpedia-af:Tim Berners-Lee`
- `dbpedia-als:Tim Berners-Lee`
- `http://am.dbpedia.org/resource/ቲም_በርነርስ_ሊ`
- `dbpedia-an:Tim Berners-Lee`
- `dbpedia-ar:Tim Berners-Lee`
- `http://arz.dbpedia.org/resource/تيم بيرنرز لى`
- `http://ast.dbpedia.org/resource/Tim_Berners-Lee`
- `dbpedia-az:Tim Berners-Lee`
- `http://azb.dbpedia.org/resource/تيم بيرنرز لى`
- `http://ba.dbpedia.org/resource/Тим_Бернерс-Ли`

FIGURE 2.4: Several types of the URI `http://dbpedia.org/resource/Tim_Berners-Lee`



The screenshot shows the DBpedia interface for the URI `http://dbpedia.org/resource/Tim_Berners-Lee`. The page displays the DBpedia logo and navigation options. Under the `rdf:type` property, a list of 16 types is provided, including:

- `owl:Thing`
- `foaf:Person`
- `dbo:Person`
- `yago:WikicatAlumniOfTheQueen'sCollegeOxford`
- `schema:Person`
- `dbo:Animal`
- `dbo:Eukaryote`
- `dbo:Species`
- `dul:NaturalPerson`
- `wikidata:Q19088`
- `wikidata:Q215627`
- `wikidata:Q5`
- `wikidata:Q729`
- `yago:WikicatInternetPioneers`
- `yago:WikicatJapanPrizeLaureates`
- `yago:WikicatPeopleFromLondon`

2.1.1 Linked Open Data

Linked Open Data (LOD), in simple words, refers to Linked Data published with an Open License. If a piece of data is published using RDF, exposed in a SPARQL endpoint, and linked with other sources, but it is private or published with a license that restricts its usage, then the complete vision of the Web of Data cannot be achieved. Semantic Web principles and technologies propose a way to publish data, but the goal of the Semantic Web it is to let the Worldwide Web community to take advantage of the possibilities of publishing data in such way. If the data cannot be used by the community, then it is not relevant for the community how it is published.

In order to encourage people to publish their datasets as LOD, Tim Berners-Lee developed a star rating model in order to evaluate the quality of the Data. The levels of this rating star system are defined as follows [40]:

★ Available on the Web (whatever format) but with an open license, to be Open Data.

★★ Available as machine-readable structured data (e.g., excel instead of image scan of a table).

★★★ As level 2, plus non-proprietary format (e.g., CSV instead of excel).

★★★★ All the above, plus use open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your stuff.

★★★★★ All the above, plus: Link your data to other people's data to provide context.

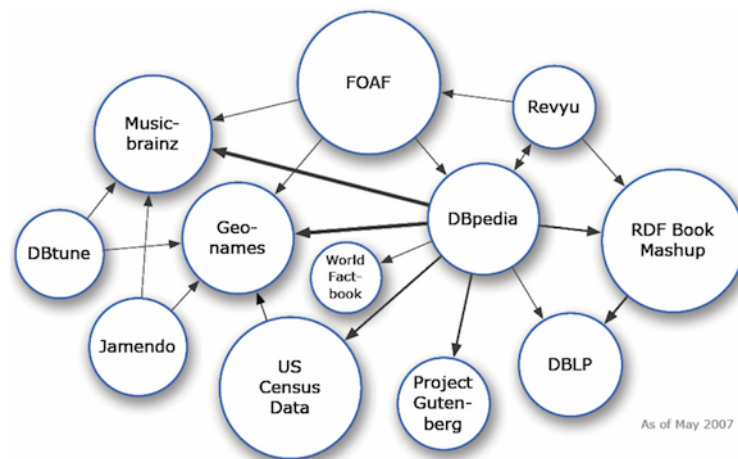
As one can see, the rating system proposed requires that the Data should be *open* to get a star. According to this system, a piece of data contained in an image could have a higher quality than an RDF dataset exposed in a SPARQL endpoint, in case this endpoint does not have an Open License. The levels defined after the first star describe scenarios of increasing data quality. The last and best level refers to LOD content.

Note that an Open License does not necessarily imply a Free License⁹. An example of frequent Open License that does indicate some restriction over the published data is CC BY¹⁰. Open licenses ensure that the published data can be reused at no cost and without asking for permission to the authors, as long as some possible conditions licenses are met. These conditions usually refer to the following questions:

- **Attributions.** The publishers may ask for credit when their work is used.
- **Commercial use.** The publishers may require that the work can be used only for non-profit purposes.
- **Derivatives.** The publishers could specify that their work can be used, but not modified.

⁹CC0 is the public domain License. Any artifact published with this license can be used by anyone with absolute no restrictions. See more at <https://creativecommons.org/share-your-work/public-domain/cc0/> Accessed in 2022/05/03

¹⁰The CC BY license allows reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator. The license allows for commercial use. Source: <https://creativecommons.org/about/ccllicenses/> Accessed in 2022/05/03

FIGURE 2.5: LOD Cloud, May 2007. Image credits to lod-cloud.net

- **Share alike.** The publishers could specify that their work can be freely used as a piece of some other works, as long as these works are shared with the same license as the publishers' one.

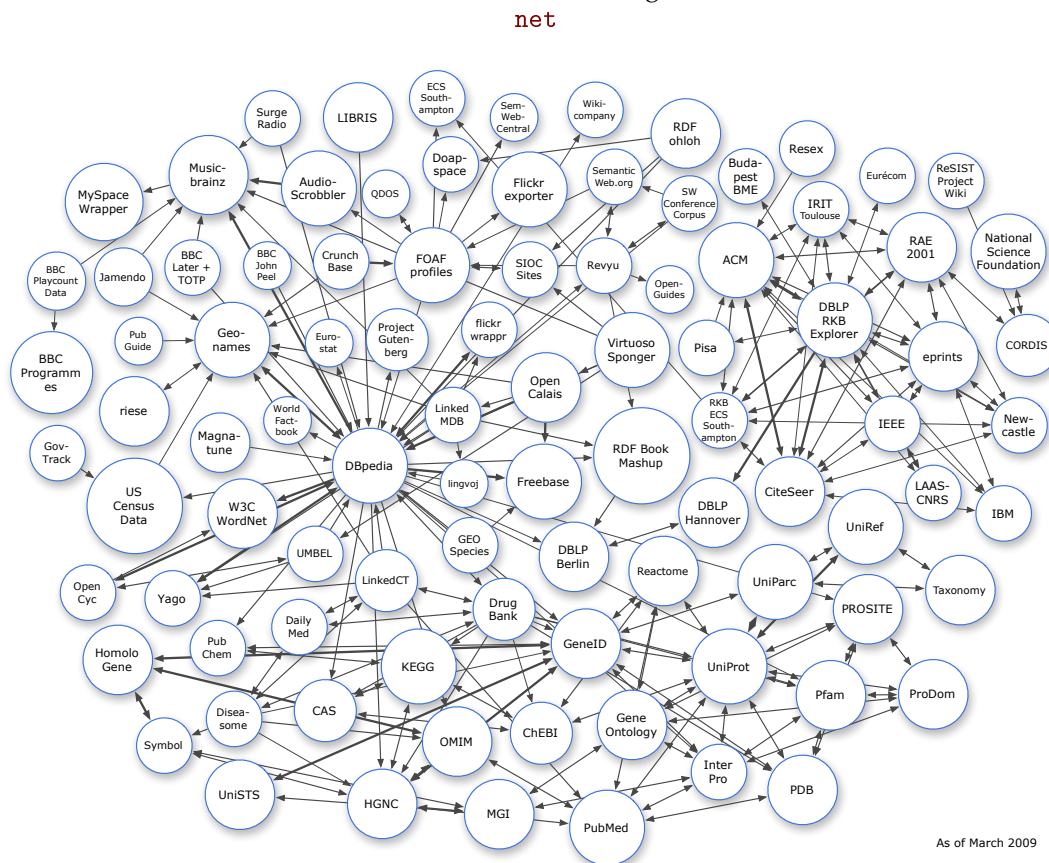
LOD projects are especially relevant for the Semantic Web community, as they are the crystallization of all the principles originally suggested to create a Web of Data. The LOD Cloud is an initiative started in 2007 aiming to represent in a map every existing LOD source and the links between these sources. The first version of the LOD Cloud, published in 2007 (one year after the definition of the LD principles) is shown in Figure 2.5. Back then it consisted of 12 projects. Just two years later, the LOD Cloud grew to the state shown in Figure 2.6. At that moment, 93 projects joined the initiative, and it was becoming hard to depict them all in a single image.

At the time of this writing, the last version of the LOD Cloud describe the relation between 1301 different projects. As one can see in image 2.7, the visual aspect of all these projects together looks like an actual cloud formed by tiny drops rather than a graph. The data volume published by these initiatives has exponentially grown too. The following list contains some insightful projects of general interest for the LOD community:

- **DBpedia** [27] is part of the LOD Cloud from its foundation and is a key project for the LOD community. Many LOD projects link to DBpedia URIs, so DBpedia can be used as a bridge among those sources. DBpedia knowledge is mainly obtained by mining Wikipedia. At the time of this writing, its core release, which is just around 14% of the total content of DBpedia¹¹, contains approximately 1 billion triples. It also contains 62 million triples linking its URIs with other elements in LOD sources.
- **Wikidata** is a collaborative project. Users from a worldwide community add and maintain the knowledge of this project with manual edits or helped by community-developed bots to build a cross-domain data source. At the time of this writing, Wikidata contains more than 1.2 billion statements about more than 90 million different entities¹². The volume of actual RDF triples is even

¹¹<https://www.dbpedia.org/blog/snapshot-2021-12-release/#anchor1> Accessed in 2022/05/03.

¹²<https://wikidata-todo.toolforge.org/stats.php> Accessed in 2022/05/03.

FIGURE 2.6: LOD Cloud, March 2009. Image credits to [lod-cloud.net](#)

bigger, as Wikidata uses a data model that allows for describing a statement (a relation between two elements) with some references or qualifiers. That is, a single relation can generate several RDF triples¹³ in Wikidata.

The evolution of the LOD Cloud in these few years reveals the increasing interest of the Web community on LOD data. LOD projects, and especially DBpedia and Wikidata, will be used in several stages of this thesis.

2.2 RDF

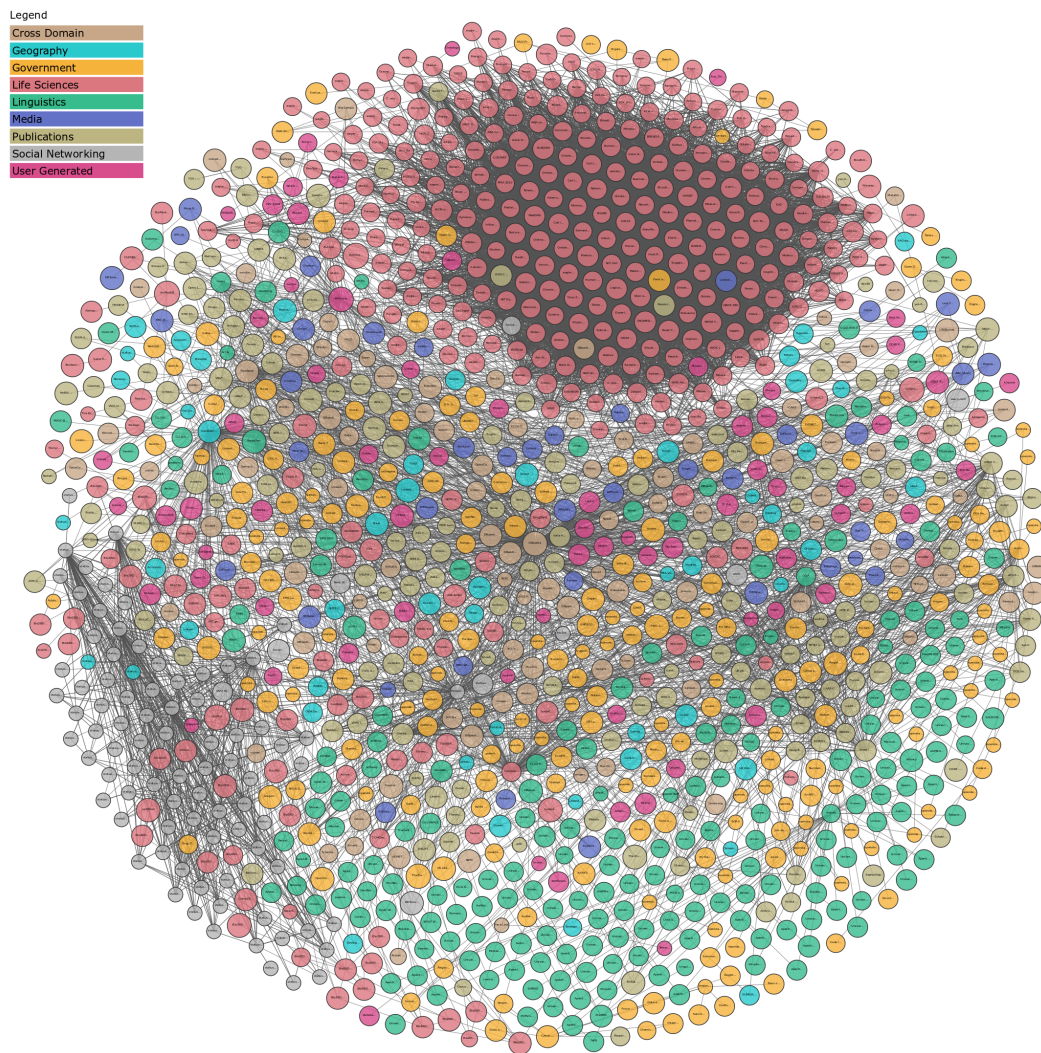
RDF is described by the W3C as “a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schemata differ, and it specifically supports the evolution of schemata over time without requiring all the data consumers to be changed” [41].

RDF is the de facto standard to produce Linked Data. It provides a simple but yet powerful mechanism to express knowledge by means of triples. Triples can be serialized in several machine-readable syntaxes built on top of the RDF data model. Its simplicity and lack of enforcing schemata allows for an easy evolution and integration of data sources.

As already stated, the minimal piece of information in RDF is the triple, which is the union of a subject, a predicate, and an object. There are three types of conceptual elements that can be used in a triple.

¹³Wikidata’s data model will be described in detail in section 3.3.2.2.

FIGURE 2.7: LOD Cloud, May 2021. Image credits to lod-cloud.net



The Linked Open Data Cloud from lod-cloud.net



- **IRIs.** The concept of URI has already been described in section 2.1. Internationalized Resource Identifiers (IRIs) are a superset of URIs. IRIs and URIs are conceptually identical, except that IRIs permit a wider range of Unicode characters. There are occasions in which operations that are just defined for URIs need to be applied to IRIs. The most insightful example of such operations is content retrieval via HTTP protocol. In such cases, IRIs should be encoded to use only ASCII characters¹⁴. For this reason, it is common to talk about URIs instead of IRIs when referring to the possible elements that can be used in RDF triples.
- **Literals.** Literals are used to represent primitive values. Strings, numbers, dates, or geographical coordinates are examples of literals. An RDF literal is composed by two mandatory elements and a third one in some special cases:
 1. **Lexical form:** It is a string representing the raw literal value.
 2. **Datatype IRI**¹⁵: It is an IRI which specifies the type of the literal, i.e., how the raw value should be interpreted. For example, the lexical form '2' could be declared, among other options, as a *string*¹⁶ or as an *integer*¹⁷, and thus be processed differently.
 3. **Language tag.** If the datatype IRI is *langString*¹⁸, then the literal can have a language tag¹⁹, i.e., an annotation describing the language in which the string's content is written.
- **Blank nodes.** Blank nodes (BNodes) are graph elements disjoint from Literals and IRIs. Except for that, the set of blank nodes is arbitrary. Unlike URIs, one cannot reference blank nodes of other RDF sources. Usually, BNodes are instruments to link elements by means of artificial structures with no actual identity. Some RDF syntaxes use identifiers to declare and reference BNodes within an RDF document. However, this artificial identity has no consistency nor meaning out of the document's context.

An RFD graph can be formally defined as a set of triples $(s, p, o) \in (\mathcal{U} \cup \mathcal{B}) \times (\mathcal{U}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$, where (\mathcal{U}) denotes all possible IRIs, (\mathcal{L}) denotes all possible literals, and (\mathcal{B}) denotes all possible blank nodes.

A triple expresses that a subject (s) is related with an object (o), with the named relation used as predicate (p). Note that this model implies that an IRI can be used in the roles of s , p , and o . This means that URIs used as named relations can be eventually used as subjects or objects too.

As one can see, the RDF model is quite simple. It essentially describes how to combine in triples three basic easy to understand concepts. The simplicity and flexibility of RDF's base model makes it suitable to be managed by web developers with lack of prior background on this topic [42].

¹⁴How to perform mapping of non-ASCII characters is explained in <https://datatracker.ietf.org/doc/html/rfc3987#section-3.1> Accessed in 2022/05/03.

¹⁵Despite literals are always compose by these elements, some RDF syntaxes have mechanism to avoid explicit type declarations.

¹⁶IRI for string type: <http://www.w3.org/2001/XMLSchema#string>.

¹⁷IRI for integer type: <http://www.w3.org/2001/XMLSchema#integer>.

¹⁸IRI for langString type: <http://www.w3.org/1999/02/22-rdf-syntax-ns#langString>.

¹⁹The language tag's syntax is described in <https://www.rfc-editor.org/info/bcp47> Accessed in 2022/05/03.

FIGURE 2.8: Example of RDF/XML syntax

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns
3  # "
4  xmlns:ns0="http://example.org/">
5
6  <rdf:Description rdf:about="http://example.org/A">
7  <ns0:knows rdf:resource="http://example.org/A"/>
8  </rdf:Description>
9
10 </rdf:RDF>
11

```

2.2.1 Different syntaxes

RDF proposes an abstract model that can be serialized using several different standard syntaxes. When RDF was proposed, eXtended Markup Language (XML) was a popular structured language for data representation among web developers. Probably for this reason, the original syntax to serialize RDF is known as *RDF/XML* and uses XML to represent the triples. The content in Figure 2.8, represents the triple (`<http://example.org/A>`, `<http://example.org/knows>`, `<http://example.org/B>`).

RDF/XML did not become too popular among semantic web developers though. First, its verbose syntax makes it difficult to write RDF content by hand. Second, the hierarchical tree model of XML differs from the graph-like RDF model, which creates some issues to manipulate RDF content with general-purpose XML mechanisms. The very same RDF document can be serialized using different XML trees, which causes difficulties to process RDF with XML standard technologies such as XPath or eXtensible Stylesheet Language Transformations (XSLT) [42].

Some other syntaxes to represent RDF content soon emerged. The main ones are probably *notation 3* (N3) [43], *turtle* [44], *n-triples* (NT) [45], and *JSON-LD* [46]. JSON-LD is defined over JavaScript Object Notation (JSON), i.e., similarly to XML/RDF, it is an approach built on top of a popular syntax among web developers which is based in a hierarchical-tree model. The other three proposals handle the concept of triple in a more natural way, and they are tightly related: turtle is a subset of N3 and, N-TRIPLES is a subset of turtle.

From this point, every example of RDF content would be written using turtle. Although this document is not meant to be a complete description of the turtle syntax, we are going to make an overview of its main aspects, so a new reader to this topic can understand the rest of the document. The complete specification of turtle is available on-line²⁰.

Figure 2.9 contains a sample piece of turtle content. Some of the main features of turtle are:

- It supports some directives to allow writing shorter URIs. These directives are the following ones:

²⁰<https://www.w3.org/TR/turtle/#BNodes> Accessed in 2022/05/03.

FIGURE 2.9: Basic example of turtle syntax

```

1
2 @prefix : <http://example.org> .
3 @prefix foaf: <http://xmlns.com/foaf/0.1/> .
4 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
5 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
6 @base <http://weso.org/example/relative/> .
7
8 #- <#Jimmy>, relative IRI
9 #- foaf:Person and rdf:type, prefixed name
10 #- a == rdf:type (special token)
11 <#Jimmy> rdf:tpe:Person .
12
13 #- a == rdf:type (special token)
14 <#Ana> a foaf:Person .
15
16 #- Predicate list
17 <#Laura> a <http://xmlns.com/foaf/0.1/ Person> ;
18     :name "Laura" ;           #- string
19     :surname "Smith"^^xsd:string ;   #- same type
20     :motto "Live and let live"@en ;   #- langString
21     :age 40 .                 #- integer
22
23 #- Object list
24 <#Ana> :likes <#Jimmy> ,
25             <#Laura> .
26
27 #- With this base, this subject == <#Jimmy>
28 <http://weso.org/example/relative/Jimmy> :likes <#Ana> .
29
30 #- Blank node with label
31 _:1 a :Unknown ;
32     :name "Not Known" .
33
34 #- Blank node with no label.
35 [ a :Unknown ;
36   :name "Not Known"
37 ] .
38
39 #- Nested unlabeled blank nodes
40 [ a :Unknown ] :likes [ a :Unknown
41                       :name "Not Known" ] .
42
43 #- List example
44 _:2 :likes _:l1 .
45 _:l1 rdf:first <#Jimmy> ;
46     rdf:rest _:l2 .
47 _:l2 rdf:first <#Ana> ;
48     rdf:rest _:l3 .
49 _:l3 rdf:first <#Laura> ;
50     rdf:rest rdf:nil .
51
52 #- Equivalent to the previous list example
53 _:2 :likes ( <#Jimmy> <#Ana> <#Laura> ) .
54

```

- **@prefix**. It associates a prefix with a namespace. It consists of four tokens: 1) `@prefix` or `PREFIX`²¹, 2) the prefix, 3) the character ‘:’, and 4) the namespace between ‘<’ and ‘>’ characters. The prefix can be empty.
 - **@base**. It declares a base namespace for the document. It consists of two tokens: 1) `@base` or `BASE`, and 2) the namespace between ‘<’ and ‘>’ characters.
- There are three different ways to write an URI:
 - **Absolute IRIs**. The whole URI is written between ‘<’ and ‘>’ characters.
 - **Relative IRIs**. The last part of the IRI is written between ‘<#’ and ‘>’. The IRI is interpreted as a concatenation of the document’s base namespace and the content between ‘<#’ and ‘>’.
 - **Prefixed names**. They consist of a prefix, the character ‘:’ and the last part of the IRI. The IRI is interpreted as a concatenation of the namespace associated to the prefix used and the content after the character ‘:’.
 - The token `a` is always interpreted as the property `rdf:type`.
 - Triples are represented as a sequence of three consecutive elements: subject, predicate, and object. Unless predicate lists or object lists are used (these two concepts are defined in the next two bullets), triples are separated from each other with a ‘.’ character.
 - When the token ‘;’ is used at the end of a triple t_i , then it is assumed that the subject of the next triple t_{i+1} is the same subject of the triple t_i . The next two tokens are expected to be the predicate and the object of t_{i+1} . This feature is called *predicate lists*.
 - When the token ‘,’ is used at the end of a triple t_i , then it is assumed that the subject and the predicate of the next triple t_{i+1} are the same subject and predicate of the triple t_i . The next token is expected to be the object of t_{i+1} . This feature is called *object lists*.
 - Literals, in general, consist of three tokens which are not separated by white spaces and appear in the following order:
 1. The lexical form between quotes.
 2. The token `^^`.
 3. The Datatype IRI.
 - There are some special types of literals whose type does not need to be declared, including the following ones:
 - In case there is just a lexical form quoted with no datatype declaration, then the literal is assumed to be `xsd:string` type.
 - Unquoted numbers are assumed to be `xsd:integer` or `xsd:float`, depending on whether they have decimals.

²¹In the most recent version of turtle, these two tokens are equivalent. Initial versions of turtle enforced the use of `@prefix`. C.f. <https://www.w3.org/TR/turtle/> Accessed in 2022/05/03.

- The unquoted words `true` and `false` are assumed to be `xsd:boolean` type.
- A quoted string followed by the `@` token and a language tag is assumed to be `rdflangString` type.
- In-line comments are supported. Everything between a `#` character and a line jump is considered a comment, except when `#` is found within an URI or a literal declaration.
- There are two ways to declare blank nodes:
 - **Labeled BNodes.** They are similar to prefixed names, except the prefix is always the token `'_'`. The identifier can be used to reference the defined BNode within the document in which it is declared, but it has no effect out of the document's context.
 - **Unlabeled BNodes.** Content between square brackets `[]` represents triples whose subject is the same blank node. Triples defined between square brackets are a succession of two elements instead of three, as the subject of them all is the BNode being defined. Predicate lists and objects lists can be used to declare several triples within the square brackets.
- Unlabeled BNodes can be nested. An example of this is shown in lines 40 and 41 of Figure 2.9.
- RDF is a model purely based in triples, but it allows to emulate lists by using some special properties, such as `rdfl:first` and `rdfl:rest`²². An example of such a conceptual list composed by the nodes `<#Jimmy>`, `<#Ana>`, and `<#Laura>` is shown in lines 44 to 50. Turtle proposes a shorter syntax to define such lists by placing a succession of elements between squares. An example of such syntax is shown in line 53. Lists written with this syntax can be used in the positions of subject or object in a triple.

In general, in the interest of brevity, we will avoid making explicit prefix declarations for the rest of turtle examples in this document. In Appendix A, we declare the namespace corresponding to any prefix used in this document, both in turtle examples and textual explanations.

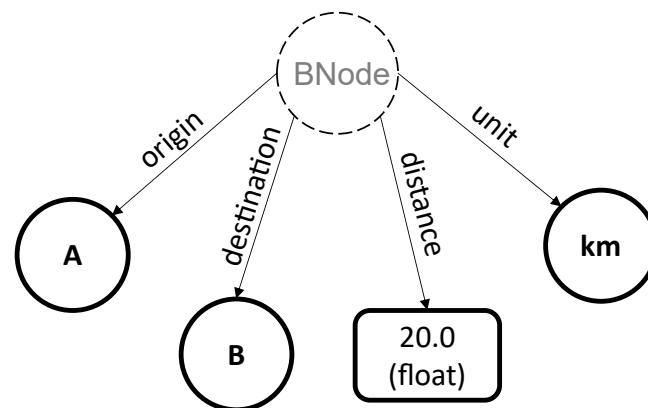
2.2.2 Extending the expressiveness of basic triples

The simple model of a triple allows for representing many kinds of factual knowledge. Nevertheless, for some scenarios, these three elements are not enough to represent a certain notion. Let us suppose that one wants to express in RDF that the distance between a city `ex:A` and a City `ex:B` is 20 km. Such piece of knowledge could be represented by a triple such as `(ex:A , ex:20KmDistnace , ex:B)`.

This representation strategy is hardly scalable. Introducing the actual distance between points in the predicate URI itself would require to define potentially infinite properties to represent every possible distance between two points. It will also prevent us from using any kind of algebra or logical operation over this quantity, as URIs are mere identifiers. Operations such as comparison, addition, etc., are defined over literals.

²²<https://www.w3.org/TR/rdf11-mt/#rdf-collections> Accessed in 2022/05/03.

FIGURE 2.10: Reification example: expressing the notion “A is 20 km far from B”



It sounds feasible to represent the actual distance using a literal, such as the floating number 20.0. However, if this literal is used as object, then the city B could not be used in that position of the triple.

RDF can solve this kind of scenarios using *reification*. Reification extends the expressiveness of RDF by means of BNodes. The RDF specification includes some standard vocabulary to implement reification schemata²³. Core elements of this vocabulary are the class *rdf:Statement*, which can be used to indicate that a certain node represents a triple, and the properties *rdf:subject*, *rdf:predicate* and *rdf:object*, which can be used to indicate which are the *subject*, *predicate*, and *object* of a *rdf:Statement* node. Nevertheless, this section of the RDF specification is non-normative, and different strategies and vocabularies to implement reification can be used [47].

An example of a simple reified model to represent the relation between the cities A and B is shown in Figure 2.10. Instead of using a direct link between A and B, both cities are linked with a BNode. The only purpose of this node is being a nexus between these two cities and the actual distance between them. Some other notions, such as the unit measure expressed by the raw value ‘20’ can be linked to the BNode too.

Reification models are not the only mechanism to increase the expressiveness of basic triples. Named graphs [48], are a standard extension of RDF which consist of turning triples into quads, i.e., a relation between four elements. This new fourth element is placed after the object and is used to associate the triple to a specific graph. Quads do not modify the basic RDF graph model, but allow for separating triples into different graphs within a single RDF document. Quads can be formally defined as $(s, p, o, g) \in (\mathcal{U} \cup \mathcal{B}) \times (\mathcal{U}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}) \times \mathcal{U}_g$, where \mathcal{U}_g denotes the subset of \mathcal{U} representing graphs.

2.2.3 ‘Schemaless’ character

RDF is often called *schemaless*, and this quality is often misunderstood. W3C’s definition indicates that RDF facilitates the integration of different sources even if they have different schemata. As stated in section 2.1, this integration is articulated by using triples stating links between different sources. The RDF core does not enforce users to use strict schemata to represent their data, as long as it is expressed

²³https://www.w3.org/TR/rdf-schema/#ch_reificationvocab Accessed in 2022/05/03.

with valid triples. This means that RDF sources have flexible and easy to integrate schemata, but it does not imply that these sources do not have internal schemata. In this context, the term *schema* is used to refer to a set of rules one may have to stick to in order to express information in a homogeneous way. Some examples of such rules in a hypothetical context could be the following ones:

- To indicate that a certain entity is instance of a certain class, the property *rdf:type* should be used.
- Every entity which represents a Person is expected to have, as most, one biological mother.
- A Person's name should be specified using the property *foaf:name*.

Using consistent schemata allow users to make an effective use of RDF content. For instance, when writing a SPARQL query to find every instance of a certain class, the user hopes that every instance-class relation is expressed with the same property. Arbitrary decision w.r.t. the instantiation property end up in pieces of knowledge being ignored due to schema incoherence. Also, when building applications over existent RDF data, the developers must know which are the data premises they can rely on. A user interface could be built under the assumption that a certain person can have a single biological mother, or that his name will be find in a triple whose predicate is *foaf:name*. Inconsistencies w.r.t. those assumptions may cause unexpected errors or extra development costs.

More on how to define and validate schemata in RDF graphs will be explained in section 2.6.

2.3 Knowledge Graphs

The term *Knowledge Graph* (KG) was early used in the literature [49]. However, this concept becomes more popular when the announcement of Google's Knowledge Graph [50]. Since then, KGs are attracting many attention from both industry and academia.

Some Linked Data sources already described or mentioned in the previous sections are KGs, including Wikidata and DBpedia. This is the case of many other well-known Linked Data sources, such as YAGO [38], or Freebase [34]. However, there is still no agreement on how to precisely define the concept of KG. Several authors have provided a definition, sometimes contradictory with existing ones [51–53]. In this document, we will use the definition of Knowledge Graph provided in a recent book survey about this topic. A Knowledge Graph is “a graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent relations between these entities” [54].

RDF proposals fit, indeed, in this definition. RDF is a graph-based model whose edges are named relations representing different types of relations between nodes. However, RDF is not the only model compatible with this generic definition. Data represented using different models of graph-based representation can be considered a KG. In [54], several popular in practice graph models are introduced:

- **Directed edge-labeled graphs.** They are defined as a set of nodes connected between them by labeled edges. RDF fits in this definition even if it introduces extra restrictions and extra proposals. In RDF, not every node can be the beginning of a directed edge, as literal nodes can be just used as objects. Also,

the fact that URIs are used to label edges allow that these labels can be used as nodes which are source or destination of other labeled edges.

- **Heterogeneous graphs.** They are also defined as a set of nodes connected by labeled directed edges, but nodes in this kind of graphs are always typed. Unlike RDF (and, in general, directed edge-labeled graphs), this type is not represented with an extra relation, but it is a fundamental part of the node itself. Heterogeneous graphs usually support clustering nodes by type. The base model of these graphs cause that each node is usually single typed. Extra types for a node can just be defined by using relations which are external to the node type.

For example, the DBLP project²⁴, which gathers information about Computer Science bibliography, is processed in [55] as an heterogeneous graph. DBLP node are classified according to types such as *author*, *journal*, *conference*, or *publication*.

- **Property graphs.** Property graphs are heterogeneous graphs which let associate key-value pairs to both nodes and edges. Property graphs are popular in graph databases, such as Neo4j [56]. The data model of property graphs is more flexible than directed edge-labeled graphs, while directed edge-labelled graphs are conceptually simpler than property graphs. However, information represented using property graphs can be mapped to directed edge-labeled graphs[47, 57]. For example, property graphs can be converted to RDF by:

1. Turning each node key-value pairs into triples. The subject would be the node itself. The key should be mapped to an URI and URI and as predicate. The value should be mapped to a literal and used as object.
2. Representing the key-value pairs attached to edges using some reification schema, as the one shown in Figure 2.10.

- **Graph datasets.** Graph Datasets are an extension of directed edge-labeled graphs that allow to handle several graphs instead of a single monolithic source. There could be several motivations for this idea. For example, one may want to associate certain information with its origin, so it is possible to differentiate between trustworthy and not trustworthy sources. One could also want to maintain different versions of a domain with frequent changes in time on separate graphs, so the evolution of the information can be traced. In RDF, graph datasets are implemented using quads.

The techniques and results described in this document are defined for RDF content. However, most of the techniques proposed or evaluated could be adapted to work with different KG data models. Particularly:

- The metrics about class importance developed or evaluated during chapter 3 could be trivially adapted to every model graph described in this section. Most of the techniques evaluated consist in general graph theory algorithms that are already defined for any network structure. On the other hand, those approaches that use RDF specific features are mainly focused on notions of instantiation expressed by means of *rdf:type* or equivalent properties²⁵. The

²⁴<https://dblp.uni-trier.de> Accessed in 2022/05/03.

²⁵Among the evaluated techniques, just ClassRank can use properties different than instantiation ones. However, ClassRank proposals can be still extrapolated to other graph models, as long as this model uses labeled edges.

notion of type is present in every KG base model described, so the proposals based on this relation could be used in other type of KG by adapting the way in which the type of a given node is found.

- The main product of chapter 4 is an automatic way to produce shapes by mining RDF content. The shape languages are defined to work just with RDF. However, the need of validating and documenting graph structures is transversal to any kind of KG. In this sense, the idea of serializing graph patterns frequently observed among groups of individuals by means of a reusable syntax can be extrapolated to other KG models. This could help in every case to reduce the cost of producing such content in a handcrafted and domain expert-dependent way.
- Again, the products of 5 are RDF shapes. However, in most cases, the value of these shapes is associated with the products that can be automatically generated with them, such as forms, text summaries, classifications, and so on. Anyhow, for users interested in the shapes themselves rather than their potential uses, the ideas exposed in this chapter could be adapted to other graph models too, as long as there are actual implementations of software pieces that can be assembled to produce a system with the architecture described in chapter 5.

KGs not always based on RDF are gaining momentum in the last years, both as open-source initiatives or private data. Just to name a few examples, companies such as eBay [58], Facebook [59], Amazon [60], Microsoft [61], and Uber [62] maintain and use their own KGs. Although this thesis is framed in the context of RDF, adapting our proposals to other popular models such as graph databases maintained with Neo4J is identified as a line of future work.

2.4 SPARQL

In the early years of semantic technologies, the need of querying RDF content emerged soon. Many languages were proposed for such a task, including RQL [63], TRIPLE [64], Xcertp [65], or Algae [66]. SPARQL [67] became the standard language to perform queries on RDF content.

Essentially, SPARQL is a graph-matching query language. The evaluation complexity of SPARQL is PSPACE-complete in case of using unlimited nested optional patterns. Nevertheless, there is a set of frequent graph patterns that avoid using certain kinds of interactions between variables in optional patterns. The graph patterns fitting in this condition are called *well-designed patterns*. SPARQL queries using well-designed patterns can be evaluated as a coNP-complete problem [67].

A SPARQL query consists of three fundamental parts:

1. **Pattern matching.** SPARQL allows for defining graph patterns. The base of these patterns are triples written in a syntax similar to turtle, where some of the elements in a triple are variables. SPARQL also provides several operators for defining such schemata, including triple nesting, union of patterns, or usage of optional patterns.
2. **Solution modifiers.** Once the results have been computed, i.e., once the subgraphs of the target RDF that match the patterns have been found, SPARQL provides several operators to filter the results or to perform some post-computation over them. Some of these operators allow for limiting the number of results,

FIGURE 2.11: Basic example of SPARQL

```

1
2 SELECT DISTINCT
3   ?person ?name
4 WHERE {
5   ?person a :Person ;
6           :occupation ?occupation .
7   OPTIONAL {?person foaf:name ?name}
8 }
9 LIMIT 50
10

```

grouping or sorting them according to some criteria, or excluding repeated values.

3. **Output.** SPARQL queries can be solved in several manners. They can consist on YES/NO questions when the keyword **ASK** is used, graph bindings to variables when **SELECT** is used, or actual RDF content when **CONSTRUCT** or **DESCRIBE** are used.

In Figure 2.11, we show an example of SPARQL query²⁶. Informally, this query could be translated to the following sentence: “Find instances of the class `:Person` that has at least an occupation and, in case they have a name, retrieve it. Also, avoid repeated results and do not show more than 50 results”. The results are retrieved in a tabular way, where the column headings are `?person` and `?name`.

The core part of a SPARQL query is the pattern matching, which consists of lines 4 to 8 in Figure 2.11. Lines 5 and 6 specify a couple of triples in turtle. However, as one can see, pieces of those triples can be substituted by variables, which are tokens starting with the ‘?’ character. These variables are the elements to match when computing the graph pattern. The query will retrieve a set of subgraphs of the target graph G containing URIs such that when the variables are replaced by those URIs, the described triples exist in G . Line 7 specifies an optional pattern. Optional patterns are used to provide extra information in the results when it is possible. Let us say there is a URI u that matches the pattern described in lines 5 and 6 in the role of `?person`. The optional pattern declared in line 7 causes that, if it exists a triple $(u, foaf:name, ?name)$ in G , then the name will be included in the results. In case this triple does not exist, then the results will include u anyway, although the name associated to u will be empty.

Line 9 allow to limit the maximum number of row results to 50. The keyword **DISTINCT** ensures that there is not any repeated row in the table retrieved. The **SELECT** keyword indicates that the result should be retrieved in a tabular way. Finally, line 3 indicates the headings (and their order) of the tabular result.

SPARQL is a complex language with many more features than the ones shown in Figure 2.11. This document does not aim to be a complete description of SPARQL, but just a brief introduction that allow the reader to 1) understand the purpose and importance of SPARQL in the RDF ecosystem, and 2) understand the SPARQL snippets or references that will be used in different chapters. The complete specification of SPARQL is available on-line²⁷, as well as many public tutorials.

²⁶As with turtle, SPARQL also accepts declaration of namespace alias. The prefixes used in this figure can be solved using Appendix A.

²⁷<https://www.w3.org/TR/sparql11-query/> Accessed in 2022/05/03.

2.5 Ontology

Ontologies are another essential piece within the RDF ecosystem. An ontology is defined as “*a specification of a representational vocabulary for a shared domain of discourse (definitions of classes, relations, functions, and other objects)*” [68]. In few words, ontologies are specifications of conceptualizations. They allow data practitioners to define vocabularies to model the universe of discourse.

In the RDF context, in short, ontologies are used to define classes, individuals, and possible relations between different nodes. Some of the notions that can be used to describe these three concepts are:

- **Classes.** A textual description in natural language can be used to define the class. They can be part of a hierarchy, i.e., they can be specializations of more general concepts and can be specialized by more specific concepts. They can also be declared as equivalent to other classes or disjoint with other classes. When two classes c_1 and c_2 are disjoint, it means that an instance of c_1 cannot be instance of c_2 too. There are some other operations available to define classes w.r.t. other classes, such as union, intersection, and complement. Also, they can define different restrictions w.r.t. property usage. These restrictions affect the type of properties that an instance of a certain class can be linked with, the number of triples with a certain predicate that the instance is expected to have, or even some features about the content expected to be found at the opposite end of such triples.
- **Properties.** They can be defined as data properties or object properties. The former ones are expected to be used in triples in which the object is a literal. In contrast, object properties are expected to be used in triples where the object is either a IRI or a Blank node. Same as classes, properties can have a textual definition and participate in hierarchies, i.e., they can specialize or be specialized by other properties. Also, they can define domain and range constraints. These are useful to specify what type of subjects and what type of objects are expected to be found in triples using a certain property as predicate. They can also be defined w.r.t. other properties, as they can be declared to be equivalent to other property or to have the inverse meaning of another element. Some logic can be attached to a property. For example, they can be declared to be symmetric²⁸ or transitive²⁹. They can also define global cardinalities, i.e., a specific number of times that it can be used with a certain node.
- **Individuals.** Individuals represent actual objects with identity in the universe modeled by the ontology. Any kind of knowledge could be expressed about an individual. Individuals can be assigned one or more types, i.e., they can be related to one or more classes with an instantiation property. They can be linked to other individuals, literals, or even classes by means of any property defined.

Classes and properties of RDF ontologies are defined using elements of some special ontologies. The W3C recommended ontologies for such a task are RDFS and OWL.

²⁸This means that, if there is a triple (e_1, p, e_2) , then the triple (e_2, p, e_1) can be inferred.

²⁹This means that, if there is a triple (e_1, p, e_2) , and another triple (e_2, p, e_3) then the triple (e_1, p, e_3) can be inferred.

RDFS allows for basic ontology definitions. Its core is mainly used to declare URIs as classes/properties/Literal types, to define class and property hierarchies, to specify some domain and range restrictions over properties, and to associate labels and comments to different entities (both to classes/properties or to URIs standing for instance-level elements).

OWL extends RDFS with many other properties and add logical axioms to ontologies, so it is possible to perform automatic inference on datasets using ontologies defined with OWL. There are two versions of OWL: OWL 1 [69], and OWL 2 [70]. OWL 2 extends OWL 1 in different ways, including:

- Syntactic sugar to make frequent statements easier to write.
- More expressivity.
- Extended support for datatypes.
- Some meta-modeling capabilities.
- Extended annotation capabilities.

In the interest of brevity, along this document, we will use just ‘OWL’ to refer to OWL 2.

The potential inferences that could be achieved by using the whole OWL vocabulary can lead to huge computational tradeoffs. For this reason, OWL defines several subsets of the language. Practitioners can decide which subset they want to use when defining a new ontology w.r.t the expected usage of this ontology. Environments in which it is important to perform automatic reasoning may use restricted subsets of OWL, so the conclusions raised by automatic reasoners can be produced in affordable time. On the other hand, ontologies which are not expected to be used in reasoning environments may not need to worry about OWL subsets and could use the whole language to achieve maximum expressivity.

The main subsets of OWL are the following ones:

- **OWL Full.** It includes the full expressivity of OWL and it is compatible with any RDF graph. However, it is undecidable, i.e., automatic reasoning cannot be applied over OWL Full.
- **OWL DL.** It represents a subset on OWL based on Description Logics. It is highly expressive and, unlike OWL Full, it does support automatic reasoning. It defines some restrictions so the graphs can be decidable³⁰. These restrictions cause that not every RDF graph can be considered an OWL DL ontology.
- **Different OWL profiles.** Like OWL DL, these profiles are based on Description Logics. However, some of the potential reasoning processes that can be performed with OWL DL are too costly in computational terms. The different profiles select subsets of OWL, so the expressivity is decreased but inference results can be obtained easier. There are three different standard profiles defined: OWL EL, OWL QL, and OWL RL.

Queries performed over ontologies defined with any of these profiles can be solved in NP-Complete³¹ complexity³².

³⁰These restrictions are specified in the following document: https://www.w3.org/TR/owl2-syntax/#Global_Restrictions_on_Axioms_in_OWL_2_DL Accessed in 2022/05/03.

³¹A problem is NP-Complete when it can be solved in Polynomial time using a Non-deterministic Turing machine.

³²https://www.w3.org/TR/owl2-profiles/#Computational_Properties Accessed in 2022/05/03.

Note that, even if RDFS and OWL are the proposed W3C vocabularies to define ontologies, it is possible to extend the classes and properties of these ontologies to create new vocabularies [71–74].

When designing an ontology, it is a good practice to care about FAIR principles [75]. FAIR is an acronym that stands for Findability, Accessibility, Interoperability, and Reusability. The FAIR principles are a set of general good practices to implement when publishing and sharing any kind of data. The scientific community identified soon the need to follow these principles for the specific case of ontology design [76]. Several authors discuss and propose specific rules or concerns to design FAIR ontologies [77–79]. These proposals could be roughly summarized in some core ideas:

- Use universal, consistent, and persistent identifiers. Both to refer to the ontology and each term defined within ontology.
- Provide complete and consistent definitions about described concepts. Specifically, link enough metadata to those concepts.
- Ensure that the concepts and the associated metadata are both machine-readable and human-understandable.
- Ensure the ontology is published with an adequate license, so it can be reused.
- Ensure the ontology is available on-line and indexed in search engines, so it can be discovered by other people.
- Design and implement reviewing and maintenance processes for the ontology.

Reusing well-designed and well-known ontologies is a key idea to maintain the Linked Data ecosystem for several reasons. By reusing existing ontologies, data producers are not forced to define a new ontology every time that they need to refer to concepts of a certain domain model. More important, using well-known ontologies causes that the data could be easily integrated with other datasets using the same vocabularies, and also makes it easier to understand for anyone who already knows the vocabularies used. There are many well-designed and widely known ontologies which are commonly used in many LOD sources. The following ones are insightful examples:

- **Dublin Core (DC)** [80]. This is a lightweight RDFS-based vocabulary to describe generic metadata.
- **Friend of a Friend (FOAF)** [81]. It is used to describe people and social relationships. It is compatible with OWL RL.
- **The Music Ontology (MO)** [82]. It is a reference ontology to describe data related to the music industry. It does not describe the music itself (notes), but concepts such as releases, albums, events, etc. It is based on OWL DL constructs, but it is not fully compatible with OWL DL reasoning nor any of its sub-profiles.
- **SNOMED CT** [83]. It is a large ontology (at the time of this writing, it includes more than 350.000 elements) that is a reference in human health domains. It describes concepts such as illnesses, symptoms, or treatments. It is fully compatible with OWL EL.

2.5.1 Assertion Box and Terminological Box

Triples in RDF sources, and, in general, any statement in a knowledge base, can be classified in two categories according to the type of elements that it describes. These categories are usually called *Terminological Box* (T-BOX) and *Assertion Box* (A-BOX) in the literature. T-BOX statements describe abstract concepts and properties. A statement such as $(:City \text{ owl:subClassOf } :HumanSettlement)$ is a T-BOX example, as it describes two elements ($:City$ and $:HumanSettlement$) which are conceptualizations, i.e., instantiable ideas that do not refer to actual individuals. On the other hand, A-Box statements contain instance information. A-BOX statements describe relations between named individuals or relations between these individuals and abstract conceptualization. The triple $(:NewYork , \text{rdf:type} , :City)$ is an A-BOX example as it links a concrete entity ($:NewYork$) with an abstract concept ($:City$). The triple $(:NewYork , :twinnedCcity , :Madrid)$ is another example of A-BOX statement. In this case, it is a link between two existing entities.

In RDF, T-BOX statements are frequently found in ontology definitions, i.e., they are used to define reusable vocabularies that model a domain of discourse. In contrast, A-BOX ones are more frequent in KGs oriented to NE-related knowledge (instances).

In this document, we frequently talk about the T-BOX and the A-BOX parts of a graph. The T-BOX part of a graph is the subgraph which is formed purely by T-BOX knowledge. Analogously, the A-BOX part of a graph is purely composed by A-BOX statements.

2.6 RDF Shapes

As explained in previous sections, ontologies can define concepts to model a knowledge domain, and they can also define restrictions about the correct use of those concepts (classes, properties) in an RDF graph. However, ontologies may not be the adequate tool to define nor validate proposed schemata in the context of a specific graph.

Let us say, for instance, that we are working with an RDF KG of a professional social network. This KG contains some personal data, such as name, surname, and age, but it also contains information about concepts related with people's jobs, such as metadata related to actual companies or occupations. Let us say that the data designers of this KG want to represent people's names using the FOAF ontology. Specifically, they want a KG with the following constraints:

- People's names will be represented with the property *foaf:name*.
- By *people*, we mean every node whose type is *foaf:Person*.
- Each person will have exactly one *foaf:name*.
- Names are supposed to be strings.
- The property *foaf:name* is expected to be used just with people. The names of other concepts (companies, job descriptions) will be specified using different properties.

If we explore the definition of the *foaf:name* property within the FOAF ontology³³, we discover that there are a couple of restrictions defined over this property:

³³http://xmlns.com/foaf/spec/#term_name Accessed in 2022/05/03.

its domain is *owl:Thing*³⁴, and its range is *rdfs:Literal*³⁵. These restrictions are compatible with the KG context, which means that the desired features of the data designers fit into the *foaf:name* formal definition. However, this definition is not enough to meet the data designers' expectations. For example, according to this definition, any node which is an instance of *owl:Thing* (or any of its subclasses) can have a *foaf:name*. Also, it does not enforce instances of *foaf:Person* to have exactly one name, and it does not enforce names to be *xsd:string* rather than any other literal type.

The actual FOAF ontology could hardly be redefined to be more specific for covering all these requirements. Doing so would imply to stop supporting many other KGs which are already using FOAF with different expectations for the *foaf:name* property. It can be feasible to define a new ontology which extends FOAF, so these restrictions are specified in the new ontology. However, this approach has several disadvantages, such as the following ones:

- Defining new well-designed ontologies is a costly task. Many times, the data practitioners using ontologies do not need nor have this skill, as they just need to develop or support the development of applications built on top of some data which is expected to meet a certain structure. These concerns are far from other questions such as the formal correction of an ontology definition nor its inference potential.
- Such new ontology would not meet FAIR principles, as it is hardly reusable out of the context of this KG. Also, it may not be a stable ontology, as changes in the schema expected for this KG may trigger changes in the ontology itself.
- A newcomer to this graph with some prior knowledge about LD would probably understand immediately the meaning of a triple using the property *foaf:name*, as FOAF is a widely used ontology. On the other hand, the same newcomer may need to learn the ontology designed ad-hoc for this case for a full understanding of the KG's content.

The task of validating the structure expected in a specific KG, or even the task of formally describing it, demands a formalism different than ontologies. Such an approach should be focused on the constraints and expectations of a specific graph context rather than trying to provide general reusable models. It should allow to express how to combine different elements of different ontologies and propose mechanisms to automatically check that the schemata defined are actually met by the target data.

Some of the earliest suggestions to that end consisted of query-based approaches. The content validation of a certain KG was performed by defining some queries exploring the graph features expected to be found. Then, those queries were executed to detect possible constraint violations. Most of those proposals were based on SPARQL, although some early approaches were not built on top of this technology. Non-SPARQL approaches include [84], which is based in the Squish query language [85], and [86], which reinterprets XPath [87] for RDF validation.

One of the most insightful SPARQL-based approach is SPARQL Inferencing Notation (SPIN) [88]. Essentially, SPIN attaches **ASK** or **CONSTRUCT** SPARQL queries to

³⁴This means that the class of every entity used as subject in a triple whose predicate is *foaf:name* should be subclass of *owl:Thing*.

³⁵This means that the object of a triple whose predicate is *foaf:name* should be a literal.

classes. The resolution of these queries can reveal some constraint violations regarding the expected structure for the instances of the class under evaluation. Some other SPARQL-based alternatives were proposed, such as [89], which combines SPARQL and SPIN; [90], which uses SPARQL and property paths³⁶; and RDFUnit [91], which describes a validation framework based on SPARQL templates.

Another family of early suggestions to solve the problem of RDF validation are the inference-based ones. These proposals are based on adaptations of RDFS or OWL to specify expected schemata and they use to rely in SPARQL to validate the structures. Inference-based proposals are affected by a couple of assumptions used in general in LD sources:

- **Open World Assumption (OWA).** LD sources may not be complete, i.e., they may not contain every available information in the world about a certain topic. OWA assumes that what is not known, it is not necessarily false. Let us say for example that one wants to list every instance of 'Q5 - human' in Wikidata. This can be achieved by executing the adequate SPARQL query in Wikidata's endpoint. This result will contain every human that Wikidata is aware of, but it will not return every existing instance of human in the world. The opposite to OWA is Closed World Assumption (CWA). CWA assumes that only the knowledge known is true.
- **Non-Unique Name Assumption.** It consists in not assuming that two different names (IRIs) always represent different concepts. The opposite to this is Unique Name Assumption (UNA). Systems working with UNA assume that different names always refer to different realities.

The use of OWA and Non-UNA decreases the validation possibilities, so most of the inference systems work under CWA and different versions of UNA. The work in [92] proposes to use OWL with CWA and a weak version of UNA to express integrity constraints. In [93], the validation problem is split in two parts: integrity constraints and closed world recognition. Description logics is used to implemented solutions by translating the axioms to SPARQL queries. Integrity Constraint Validation (ICV) [94] proposes an approach similar to these works. In ICV, constraints are written in OWL syntax, but using semantics based on CWA and UNA. These constraints are translated to SPARQL to perform the validation. ICV is part of the Stardog database³⁷.

Some novel languages were developed to solve the validation problem. OSLC Resource Shapes [95], Dublin Core Application Profiles [96], and RDF Data Descriptions [97] are examples of such languages. Nevertheless, the most relevant proposals are Shape Expressions (ShEx) [21], which is used for validation and description in some major KGs (such as Wikidata), and Shapes Constraint Language (SHACL) [20], which has become a W3C recommendation for RDF validation. These two languages are described in the following subsections.

2.6.1 Shape Expressions

ShEx is a language to describe and validate RDF sources. Its development started at 2013 and the original intention was to provide a human-readable syntax to OSLC

³⁶A property path is defined as "a possible route through a graph between two graph nodes. A trivial case is a property path of length exactly 1, which is a triple pattern. Property paths allow for more concise expression of some SPARQL basic graph patterns and also add the ability to match arbitrary length paths." Cf. <https://www.w3.org/TR/sparql11-property-paths/> Accessed in 2022/05/03.

³⁷<https://www.stardog.com/> Accessed in 2022/05/03.

FIGURE 2.12: ShEx example: a person should have a *foaf:name*.

```

1
2  :Person {
3     a [ foaf:Person ] ;
4     foaf:name xsd:string ;
5  }
6

```

FIGURE 2.13: Turtle example: a person with a *foaf:name*.

```

1
2  :jimmy a foaf:Person ;
3         foaf:name "James" ;
4         foaf:age 45 .
5

```

Resource Shapes, but it became more expressive than this language [42]. Its syntax and semantics were influenced by OSLC, turtle, and SPARQL, and it also took some ideas from regular expressions and Relax NG [98].

Graphs can be validated and described using ShEx schemata. A ShEx schema is a set of labeled shape expressions, and a labeled shape expression consist of the association of a label with a set of node constraints or shapes. These shapes and node constraints are the way to express how the neighborhood of a certain node is expected to be. Let us say we want to express the example introduced in this section using ShEx, i.e., that a node of type person should have exactly one *foaf:name*. Using ShExC syntax, this can be expressed as it is shown in Figure 2.12³⁸.

By default, ShEx shapes are *open*. This means that a node conforming to a shape must have the features described by this shape, but it can use other properties too. For instance, the node *:jimmy* described in Figure 2.13 conforms to the shape *:Person* defined in Figure 2.12. *:Jimmy* is declared to have type *foaf:Person*, and it has exactly one *foaf:name* which is a string. Also, it is the subject of a triple with *foaf:age*. However, since the shape *:Person* is open, this does not affect the validation.

A shape can be *closed* too. To do this in ShExC, one should use the keyword **CLOSED** before the definition of a shape. If we try to validate *:jimmy* against a closed version of the shape *:Person*, the node will not conform, as the usage of *foaf:age* is not defined by this shape.

Figure 2.12 provides a description of how a node associated to the label *:Person* should be like. However, it does not express in any way which are the actual nodes whose conformance with this shape should be checked. To do so, ShEx proposes to use shape maps. Shape maps are associations between node groupings and shape labels. In Figure 2.14, we show a shape map including two example associations. The first one, in line 2, indicates that the node *:jimmy* should be evaluated against the Shape *:Person*. The second one, in line 4, uses a triple pattern to indicate that every node which is an instance of *foaf:Person* should be evaluated against the shape *:Person*.

A standalone ShEx schema can be enough to describe the structures expected to be found in an RDF KG. In contrast, to perform validation, a ShEx implementation

³⁸As with turtle, ShEx allows for defining alias for namespaces. It uses a similar syntax with **PREFIX** declarations to that end. In the interest of brevity, we will avoid these declarations for every ShExC example in this document. The prefixes used in the examples can be solved using Appendix A.

FIGURE 2.14: Shape map example

```

1
2 <http://example.org/jimmy> @ <http://example.org/Person>
3
4 { FOCUS a foaf:Person } @ <http://example.org/Person>
5

```

FIGURE 2.15: ShExC example content

```

1
2 :Employee :Person AND {
3   :occupation @:Occupation * ;
4   :worksFor @:Company {1,3} ;
5   ( :primaryID xsd:string
6     |
7     :alternativeID Literal
8   )
9 }
10
11 :Person {
12   a [ foaf:Person ] ;
13   foaf:name xsd:string
14 }
15
16 :Occupation IRI AND {
17   rdfs:label rdf:langString +
18 }
19
20 :Company {
21   :denomination xsd:string ;
22   :sector [ :primary :secondary :tertiary ] ? ;
23   ^ :worksFor @:Employee + ;
24   :relatedWith IRI *
25 }
26

```

requires three different elements:

- A ShEx schema S .
- Some target RDF content T .
- A shape map M , indicating which nodes of T should be validated against which shapes of S . This input shape map is called *fixed shape map*.

The output of the validation process should be a new shape map, which associates nodes and shape labels. Those associations are qualified as either *conformant* or *non-conformant*. Non-conformant associations also include an error message explaining why the validation of the node and the shape label associated failed. This kind of shapes maps are called *result shape maps*.

In Figure 2.15, we provide a ShEx schema written in ShExC which includes some of the major features of ShEx. We will use this example to make an overview of such features:

- Shapes express features expected to be found in a *focus node*. We say a node n conforms with a shape s when the triples described by s exists if n is used as focus node.

- Shapes are composed by triple expressions. A triple expression is a set of triple constraints. A triple constraint consists of three elements:
 - **A property.** It indicates what property should be used in a triple. They are always IRIs.
 - **A node constraint.** It expresses the type of the object which is expected to be found in a triple. They can have different types of values:
 - * **Macros.** They are keywords expressing sets of possible values. For example, the macro `Literal` used in line 7 stands for any kind of literal. The macro `IRI` used in line 24 stands for any kind of IRI. Some other usual macros are `BNode`, which stands for blank nodes, or the macro `'.'`, which matches any value.
 - * **Some IRI.** They indicate the URI associated to a certain literal type. For example, `xsd:string` is used in lines 5, 13, and 21 to express that the triple described expects to have a string in the role of object.
 - * **Value sets.** Value sets are expressed as a succession of values within square brackets separated by blanks. A node constraint matches a value set when its content matches exactly one of the values within the value set. For instance, line 12 uses a value set that only contains the URI `foaf:Person`. The value set in line 22 would match the URIs `:primary`, `:secondary`, and `:tertiary`.
 - * **Shape labels.** This node kind matches nodes that conform with the shape associated with the indicated label. A shape label is preceded by the character `'@'`. See examples in lines 3, 4, and 23.
 - * **Nested shapes.** This node kind works similarly to a shape label. However, instead of using a label, one can write an actual shape between curly brackets in the node constraint position of the triple constraint. Those shapes are called *nested shapes* and cannot be referred out of the scope of the triple constraint in which they are defined, as they have no label.
 - **A cardinality.** It expresses how many times a the triple described is expected to be found. When no cardinality is specified, it is assumed that the triple described is expected to be found exactly once. The other possible cardinality values are:
 - * **Kleene closure `'*'`** (see line 3). The triple described is expected to be found between 0 and an unbounded number of times.
 - * **Positive closure `'+'`** (see line 17). The triple described is expected to be found between 1 and an unbounded number of times.
 - * **Optional cardinality `'?'`** (see line 22). The triple described is expected to be found between 0 and 1 times.
 - * **Exact range**, specified with two positive integers between curly braces (see line 4). The triple described is expected to be found a number of times in the range $[a, b]$.
- A triple constraint is inverse when it starts with the character `'^'` (see line 23). An inverse triple swaps the roles of the focus node and the node constraint. That is, the triple described expects to find the value matching the node constraint in the role of subject, and the focus node in the role of object. A triple constrains can also be negated with the character `'!'`. This means that the triple described should not be found in the focus node's neighborhood.

- Triple constraints can be joined with the *eachOf* operator, expressed with the character ‘;’. Triples joined with this operator are called *triple expression*. A node must match every triple constraint within a triple expression in order to match the triple expression.
- The logical operators *and*, *or*, and *not* are supported with its usual semantics using the keywords **AND**, **OR**, and **NOT** respectively. Complex structures can be specified using the logical operators **AND**, **OR**, and **NOT**. For example, a node matching the shape `:Employee` should conform at a time with the shape `:Person` and the triple expression described in lines 2 to 9.
- The operator *oneOf* is supported. The character ‘|’ is used to that extend. A *oneOf* operator joining several expressions (each element joined can be a triple constraint or a triple expression in brackets) indicates that the focus node should conform with just one of the expressions. For example, lines 5 to 8 indicate that an `:Employee` must have an identification. This identification can be indicated using the property `:primaryId` or the property `:alternativeId`, both a node conforming with `:Employee` cannot use both properties at a time.

The ShEx features shown in this example do not cover all the language’s possibilities. However, they are enough to understand the ShEx snippets that we will use along this document. At the time of this writing, ShEx 2.1 is the last version published by Shape Expression’s W3C Community Group and its complete specification is available on-line[99].

In this example, we have used ShExC, but ShEx can be expressed using some other syntaxes. ShExJ[100] is a sub-language of JSON [101]. ShEx can also be written in any RDF syntax by using ShExR. The RDF representation of ShEx is defined in the ShEx JSON-LD context³⁹. ShExC, ShExJ, and ShExR are equivalent. They have identical expressivity and there are complete mappings from one syntax to another.

Despite ShEx has not become a W3C recommendation, it has been adopted as the validation language for Wikidata. It is also being used in a variety of scientific works related to RDF validation or description [102–107].

There are implementations of ShEx validators in several programming languages, including JavaScript⁴⁰, Java⁴¹, Ruby⁴², Python⁴³, and Scala⁴⁴.

2.6.2 Shape Constraint Language

SHACL is a recommendation proposed by the W3C to perform validation of RDF content. SHACL has influences of SPIN, ShEx, and OSLC [42]. SHACL and ShEx are not equivalent languages: not everything expressed in SHACL can be mapped to ShEx, and vice-versa. Nevertheless, the core intention of both languages is similar: the validation of RDF sources. Also, most of the core features of both languages can be easily mapped from one to another.

SHACL is expressed in RDF. At the time of this writing, there is an ongoing draft of a Compact Syntax for SHACL [108]. However, this compact syntax is not part of the W3C recommendation, and its expressiveness does not match the expressiveness of SHACL RDF. In this document, we will use turtle to write SHACL content.

³⁹<http://www.w3.org/ns/shex.jsonld> Accessed in 2022/05/03.

⁴⁰<https://github.com/shexjs/shex.js> Accessed in 2022/05/03.

⁴¹<http://shexjava.lille.inria.fr/> Accessed in 2022/05/03.

⁴²<https://github.com/ruby-rdf/shex> Accessed in 2022/05/03.

⁴³<https://github.com/hsolbrig/PyShEx> Accessed in 2022/05/03.

⁴⁴<http://www.weso.es/shex-s/> Accessed in 2022/05/03.

Same as ShEx, SHACL is built around the concept of *shape*. SHACL distinguishes two kinds of shapes:

- **Node shapes** (*sh:nodeShape*). They are conceptually similar to ShEx shapes: they describe topological features that are expected to be observed among the triples involving a certain focus node.
- **Property shapes** (*sh:propertyShape*). They are used to describe constraints that should be met w.r.t. nodes that can be reached from the focus node by following a certain path.

A fundamental difference between SHACL and ShEx is that SHACL shapes include explicit references to the nodes they are supposed to conform with, while, when working with ShEx, one must provide an independent shape map to introduce such information in a validator. The SHACL statements that indicate which nodes should be evaluated with a certain shape are called *target declarations*. A node shape must have at least a target declaration, which can be of one of the following types:

- **Target node** *sh:targetNode*. A certain node is supposed to conform with the shape.
- **Target class** *sh:targetClass*. Every instance of a certain class is supposed to conform with the shape.
- **Subjects of a certain property** *sh:targetSubjectsOf*. Every subject of a triple whose predicate is a certain IRI is supposed to conform with the shape.
- **Objects of a certain property** *sh:targetObjectsOf*. Every object of a triple whose predicate is a certain IRI is supposed to conform with the shape.

By default, SHACL shapes are open too, i.e., a node *n* conforms with a SHACL shape *s* when *n* conforms with every constraint defined by the shape. *n* can be used in triples that are not profiled by any constraint in *s* and still be conformant with *s*. However, *s* could be declared closed using the property *sh:closed*. In this case, *n* will be conformant with *s* just in case it does not use more properties than the ones described in the constraints of *s*.

In Figure 2.16, we show a SHACL mapping of the ShEx content shown in Figure 2.15. We can use this example to review some of SHACL's major features:

- A node whose type is *sh:nodeShape* is a SHACL shape⁴⁵.
- Every shape declares a target. As this was not necessary for ShEx, this part of the example has been filled considering reasonable values for each case:
 - *:Person*. Every instance of *foaf:Person* (see line 27). By using this target class, we ensure that the triple constraint for the ShEx shape *:Person* declared in line 12 of Figure 2.15 is always fulfilled. Just instances of *foaf:Person* will be evaluated with the SHACL shape *:Person*, then it will be true for any entity *e_i* meeting this condition that it exists a triple such as $(e_i , a , foaf:Person)$.

⁴⁵This is one of the mandatory conditions that a node should met to be a well-defined SHACL shape. The rest of formal conditions can be read at <https://www.w3.org/TR/shacl/#shapes> Accessed in 2022/05/03.

FIGURE 2.16: SHACL example in turtle syntax

```

1
2 :Employee a sh:nodeShape ;
3   sh:targetSubjectsOf :worksFor ;
4   sh:node :Person ;
5   sh:property [ a sh:propertyShape ; #- Optional declaration.
6                 sh:path :occupation ;
7                 sh:node :Occupation ;
8                 ] ;
9   sh:property [ sh:path :worksFor ;
10               sh:node :Company ;
11               sh:minCount 1 ;   sh:maxCount 3 ;
12               ] ;
13   sh:xone (
14     [ sh:property [ sh:path :primaryId ;
15                   sh:datatype xsd:string ;
16                   sh:minCount 1 ;   sh:maxCount 1 ;
17                   ]
18     ]
19     [ sh:property [ sh:path :alternativeId ;
20                   sh:nodeKind sh:Literal ;
21                   sh:minCount 1 ;   sh:maxCount 1 ;
22                   ]
23     ]
24   ) .
25
26 :Person a sh:nodeShape ;
27   sh:targetClass foaf:Person ;
28   sh:property [ sh:path foaf:name ;
29               sh:datatype xsd:string ;
30               sh:minCount 1 ;   sh:maxCount 1 ;
31               ] .
32
33 :Occupation a sh:nodeShape ;
34   sh:targetObjectsOf :occupation ;
35   sh:property [ sh:path rdfs:label ;
36               sh:datatype rdf:langString ;
37               sh:minCount 1 ;
38               ] .
39
40 :Company a sh:nodeShape ;
41   sh:targetObjectsOf :worksFor ;
42   sh:nodeKind sh:IRI ;
43   sh:property [ sh:path :denomination ;
44               sh:datatype xsd:string ;
45               sh:minCount 1 ;   sh:maxCount 1 ;
46               ] ;
47   sh:property [ sh:path :sector ;
48               sh:in ( :primary :secondary :tertiary ) ;
49               sh:minCount 1 ;   sh:maxCount 1 ;
50               ] ;
51   sh:property [ sh:path [ sh:inversePath :worksFor ; ] ;
52               sh:node :Employee ;
53               sh:minCount 1 ;
54               ] ;
55   sh:property [ sh:path :relatedWith ;
56               sh:nodeKind sh:IRI
57               ] .
58

```

- *:Company*. The *:Company* ShEx shape does not declare an expected type. It seems reasonable to use the property *:worksFor* to define the targets of this shape in SHACL. By using *sh:targetObjectsFor* in line 44, we declare that any node which is an object in a triple whose predicate is *:worksFor* should be evaluated with the shape *:Company*.
 - *:Occupation*. This is a case similar to *:Company*, and we followed the same strategy. We used a target declaration with *sh:targetObjectsFor* in line 34, such that every node which is used as object in a triple whose predicate is *:occupation* will be evaluated with the shape *:Occupation*.
 - *:Employee*. Although the *:Employee* ShEx shape does not include a direct triple constraint indicating a type expected, the focus nodes evaluated are supposed to be *foaf:Person* type, as every node conforming with *:Employee* should conform with the shape *:Person* too. Then, a target declaration based on *sh:targetClass* should be avoided in this case, as the class *foaf:Person* is already associated with the shape *:Person*, and not every instance of *foaf:Person* need to conform with the shape *:Employee*. Thus, we use a target declaration based on the property *:worksFor* and *sh:targetSubjectsOf*.
- Most of the triple constraints expressed in the ShEx example are transformed to a property shape associated to a node shape with *sh:property*. A node is assumed to be a property shape when it is used as subject in a triple whose predicate is *sh:path*. Optionally, the type of property shapes can be explicitly declared as it is shown in line 5.
 - The *sh:path* of a property shape indicates a property (or a more complex path) to follow from the focus node in order to evaluate a certain constraint. Most of the examples of *sh:path* shown in Figure 2.16 indicate simple properties, but a more complex path is shown in line 51. Here, *sh:path* is linked to a SHACL *property path*. SHACL property paths are nodes which indicate complex paths from a focus node. In this case, the property path used indicates an inverse path, i.e., a path where the focus node is used as object instead of subject. Other possible types of property paths can be used⁴⁶.
 - Different SHACL properties are used to specify what type of node is expected to be found at the opposite end of the focus node after following a certain *sh:path*:
 - *sh:node*. It is used to refer to other shapes (see line 7). Mind also the use of *sh:node* in line 4. In this case, the subject of the triple with *sh:node* is a node shape instead of a property shape. In any case, *sh:node* indicates that the subject of the triple should conform with the shape indicated in the object of the triple.
 - *sh:datatype*. It is used to make reference to specific literal types (see line 15).
 - *sh:nodeKind*. It is used to make reference to certain special SHACL values that match broad node categories. In this example we have used *sh:Literal* (see line 20) and *sh:IRI* (see line 56), but more values are defined⁴⁷.

⁴⁶Cf. <https://www.w3.org/TR/shacl/#property-paths> Accessed in 2022/05/03.

⁴⁷<https://www.w3.org/TR/shacl/#NodeKindConstraintComponent> Accessed in 2022/05/03.

- *sh:in*. It is used to make reference to a fixed set of values by means of an RDF collection (see line 48).
- Cardinalities are specified with the properties *sh:minCount* and *sh:maxCount*. The default cardinality of a property shape is *zero or more*. An explicit definition of *sh:minCount* can change the minimum boundary, while a explicit definition of *sh:maxCount* can change the maximum boundary.
- The logical operators **AND**, **OR**, and **NOT** are supported with the properties *sh:and*, *sh:or*, and *sh:not* respectively.
- The operator *oneOf* shown in ShEx for this example can be emulated with the property *sh:xone*⁴⁸. The object of a triple whose predicate is *sh:xone* is expected to be an RDF collection of property shapes (see lines 13 to 24). This collection should contain one or more elements. The subject of this triple is supposed to conform with just one of the members of this collection.

This overview does not include every SHACL feature. It just covers some of its most usual constructions and it is enough to understand the SHACL examples used along this document⁴⁹.

The SHACL specification is divided in two sections. SHACL Core and SHACL SPARQL. Every feature used in the example shown in Figure 2.16 is defined within SHACL Core. SHACL SPARQL adds the capacity to define SPARQL-based constraints to the SHACL languages. SHACL Core does not have any dependency with SHACL SPARQL, so SHACL validators focused on SHACL Core can ignore SHACL SPARQL.

A SHACL validation requires two conceptual inputs that could be provided be provided with a single RDF source or as independent elements. On the one hand, they need a graph containing shape definitions such as the one shown in Figure 2.16. On the other hand, they need a data graph. This graph is supposed to contain the nodes that should be evaluated against the shapes defined. The result of the validation process is an RDF graph as well, which is called *validation report*. These reports are described using the SHACL vocabulary for validation reports⁵⁰. They have the following essential features:

- They consist of a graph with exactly one instance of *sh:ValidationReport*. Although it is not mandatory, the validation report is usually a BNode.
- The *sh:ValidationReport* is the subject of exactly one triple with the predicate *sh:conforms*. The object of this triple can be either *true* or *false*.
- If *sh:conforms* is *true*, it means that no constraint violation has been detected, so every target of every shape defined conforms with its corresponding shapes. If *sh:conforms* is *false*, it means that some constraint violations have been detected by the validator. Then, the node *sh:ValidationReport* is linked to several instances of *sh:ValidationResult* with the property *sh:result*. These instances are usually BNodes too.

⁴⁸ShEx's *oneOf* and SHACL's *sh:xone* are not fully equivalent. This will be explained in section 2.6.3.

⁴⁹The complete SHACL specification is available at <https://www.w3.org/TR/shacl/> Accessed in 2022/05/03.

⁵⁰<https://www.w3.org/TR/shacl/#dfn-validation-report> Accessed in 2022/05/03.

- Each node `sh:ValidationResult` is linked to some other properties indicating several aspects of the violation detected, such as shape, non-conformant focus node, human readable text message describing the error, value causing the constraint violation, path from the focus node, etc.

There are also some advanced SHACL features that are still not part of the standard [109]. Such advanced features are the following ones:

- **Custom targets.** The targets of a node shape are not limited to the properties already mentioned. Custom groups of nodes can be defined as the result of SPARQL SELECT queries.
- **Annotation properties.** With this feature, validation results based on SHACL SPARQL can be more informative.
- **Functions.** This mechanism allows for defining operations which produce an RDF term by processing zero or more parameters from a data graph. These operations are based in functions defined in SPARQL. There are also some SHACL JavaScript extensions [110] that are not part of the standard either.
- **Node Expressions.** They allow to refer to a set of RDF nodes w.r.t. the focus node in the context of a shape definition. This means that this set of nodes has potentially a different value for each focus node evaluated against a certain shape defining a node expression.
- **Expression constraints.** They allow to evaluate a logical operation on the nodes corresponding to a given node expression. Expression constraints are evaluated as `true` when every node of the node expression is evaluated as `true` for the logical operation defined. Otherwise, they are evaluated as `false`.
- **SHACL Rules.** They propose a mechanism to define rules within SHACL shapes that are capable of infer new RDF triples from asserted triples.

SHACL is being used in a wide variety of projects[111–116] and there are SHACL implementations for many programming languages such as Java⁵¹, Python⁵², JavaScript⁵³, Ruby⁵⁴, and Scala⁵⁵.

2.6.3 Brief comparison between SHACL and ShEx

SHACL and ShEx are not equivalent languages. Some fundamental differences between SHACL and ShEx have already been mentioned in the previous subsections, such as 1) different ways to indicate which nodes should be evaluated with which shapes; 2) different results outputted by validators; 3) different status w.r.t. W3C standardization; or 4) some syntactical and semantic differences. Here we make a list of some other key differences between ShEx and SHACL:

- **General aim of validation results: schema vs. constraints.** The usual validation results of a SHACL implementation consist of a report of detected constraint violations. When no errors have been detected, the results consist

⁵¹<https://github.com/TopQuadrant/shacl> Accessed in 2022/05/03.

⁵²<https://github.com/RDFLib/pySHACL> Accessed in 2022/05/03.

⁵³<https://github.com/TopQuadrant/shacl-js> Accessed in 2022/05/03.

⁵⁴<https://github.com/ruby-rdf/shacl> Accessed in 2022/05/03.

⁵⁵<https://github.com/weso/shaclex> Accessed in 2022/05/03.

of a single node with *sh:conforms true*. On the other hand, ShEx's shape map results include evaluation failures but also nodes successfully validated. This type of result allows for directly performing further operations with those nodes whose conformance with a certain shape has been confirmed. Of course, a similar goal could be achieved with SHACL in a scenario of no errors reported, as the positive results allow to solve the nodes conforming with the shapes by turning the target declarations of each shape in SPARQL queries. This is a slight difference between the aim of these two languages though. While SHACL's main goal is to verify that some RDF content satisfies a set of constraints, ShEx aims to report an explicit schema on the graph evaluated rather than just pure constraint violations.

- **Inference.** ShEx computes the RDF content as it is sent to the validator. No inference to produce new triples is performed at any point of the validation. SHACL, in contrast, does have some mechanisms that use inference. For instance, the `[sh:targetClass :A]` declaration refers to explicit instances of `:A`, but also to instances of any subclass of `:A`, which will be detected by using inference.
- **Violation severity.** In general terms, SHACL shows a higher granularity when reporting validation errors. Each *sh:ValidationResult* is associated to a severity level, which is one of *sh:Info*, *sh:Warning*, and *sh:Violation*. SHACL validators also provide different information about the error found. In contrast, ShEx validators do not enforce the use of fixed vocabularies to this extent. A ShEx validator must produce associations of nodes and shapes, either conformant and non-conformant, and they also must produce an informative message for the non-conformant cases. However, there is not a strict definition on the format or content of such messages.
- **Property paths.** SHACL supports property paths natively. In contrast, ShEx's triple constraints are always paths of distance 1 from the focus nodes (either direct or inverse). However, property paths can be emulated in ShEx using nested shapes.
- **Recursion.** ShEx support recursion, either direct (a shape makes a reference to himself in a triple constraint) or indirect (different shapes mention each other so they create a cyclic-dependent validation). Although some types of recursion can be described with property paths, in general, recursive shapes are not permitted in SHACL.
- **Repeated properties.** ShEx allows to define different triple constraints that have the same property. In contrast, when different property shapes with the same *sh:path* value are defined in SHACL, they behave conjunctively. This means that the constraints defined for each property shape will be applied to validate every property shape with the same *sh:path*. In SHACL, this limitation can be bypassed using qualified shapes.
- **OneOf partial matches.** ShEx's *oneOf* and SHACL's *sh:xone* behave differently when some of the expressions joined by these operators are composed by more than one triple constraint in ShEx or more than one property shape in SHACL. *OneOf* detects partial matches, i.e., a node conforming with an expression in a *oneOf* must not conform with any of the elements defined in the rest of the expressions in order to match the *OneOf* expression. In contrast,

SHACL produces a positive validation of a *sh:xone* when the focus node completely conforms with just one of the elements joined. Partial matches of other elements in the *sh:xone* does not affect the validation.

- **Annotations.** ShEx supports annotations, i.e., predicate-object pairs that can be attached to a certain construction and are represented internally as triples whose subject is that construction. In ShExC, annotations are written as in-line comments. Annotations can be used to enrich the information on user interfaces based on shapes. SHACL does not support this very same concept, but it does define some special properties such as *sh:name* and *sh:description*, which are used to attach textual information to a certain node. These properties are ignored by the SHACL validator, but they can also be used to generate richer user interfaces based on shapes.

A thorough comparison between ShEx and SHACL is provided in [42]. In the context of this thesis, the differences between ShEx and SHACL are relevant for users interested in automatic generation of shapes by means of the system described in chapter 4.

2.7 Social Media

The term *social media* did not emerge in academic environments, but in popular press [4]. However, social media attracted the interest of the scientific community very early, due to its cultural impact and its general pervasiveness. Social media sites have been mined or used for a wide variety of purposes. Such purposes include, but are not limited to, various aspects of marketing [8], user profiling [9], community detection and clustering [10], electoral predictions[11], depression and mental illnesses detection[12], suicide prevention [13], disaster detection and crisis management [14], analysis of gambling operators [15], cyber-bullying detection [16], fake news detection [17], or, quite recently, studies on the positive and negative aspects of social media usages during the COVID-19 pandemic [18].

Social media is indeed one of the fundamental pillars on which the so-called *Web 2.0*, considered an evolution of *Web 1.0*, was built. The term *Web 1.0* frames a scenario in which the web content mostly consisted of static HTML pages populated with content entirely selected by their owners. By contrast, *Web 2.0* enables multi-directional communication and web content created with user actions and interactions.

The notion on the exact nature of social media is not fully settled. The so-called *Social Network Sites* (SNS), such as Twitter or Facebook, are consensual examples of social media. However, it is unclear whether some other sites or on-line communication mechanisms should be classified as social media instances. For example, let us think about Twitch, a trendy platform in the last years. The main goal of Twitch is being a multimedia platform where creators can stream their content to be consumed by the users. Despite this, it has some fundamental features that one may intuitively assign to any social media, such as:

- Any user can become a content creator and a content consumer.
- There are mechanisms to establish multi-directional communications (chats, comments).

- There is a graph of user profiles in which one may select who to follow, i.e., who's updates must be primarily (or exclusively) shown.

In [117] a social media classification is proposed. Each social media instance is viewed as one of SNSs, collaborative projects, content communities, blogs, virtual game worlds, and virtual social worlds. Despite this classification not being normative and prone to change in the future with the emergence of new concepts or platform types, it gives some clues about the diversity of systems that can be seen as social media.

Many definitions of social media have been proposed (nearly as many as social media studies have been performed). In this document, we work with the definition of [4]: “*Social media represent a set of communication practices that can typically be described as ‘many-to-many’. In contrast to broadcast media, consumers are typically also producers. In contrast to in-person communication, audiences are often ambiguous or underspecified*”. This definition draws two fundamental features of any social media platform:

- **‘Many-to-many’ communication.** This type of communication indicates that there are no fixed roles for users as producers and consumers of content. In social media, every user can be both a producer and a consumer eventually. This is a key difference with broadcast media such as television, radio, or newspapers, where a few agents produce content that many users consume, with rare and marginal forms of native feedback.
- **Nonspecific audiences.** A key feature of a social media platform is being able to share content with unspecific recipients. Potential consumers of some content posted by a user could be every other user of the platform (public) or a reduced group of people (such as Facebook friends or Twitter followers). This feature distinguishes social media from other type of communications, such as emailing some content to a large list of users, or using messaging platforms such as WhatsApp or Telegram. Note that both the content shared on a SNS and the desired audience for that content can be the same as, for example, when someone sends an email to multiple recipients. However, the later case implies to forward the content to specific individuals, while content on SNS remains non-addressed even if some sorts of access restriction are implemented. Many social platforms implement secondary mechanisms to provide addressed communication though.

Note also that the examples provided about what is and what is not social media may not be consensual. Some authors argue that certain systems associated to the stage of Web 1.0, such as email-lists, fit in the concept of social media [5]. They argue that original Berners-Lee’s vision of the Web consisted in a collaborative reader-writer platform, but during the 1.0 stage the Web was dominated by static web pages. Then, Web 2.0 is indeed an evolution from Web 1.0, but also a reality closer to the original Berners-Lee’s idea.

Some other authors argue that any communication media is indeed a social media, as the act of communication itself is intrinsically social. Moreover, they argue that the platforms known as social media are *less social* [6] than previous types of virtual communications, as they involve private actors (owners of the platforms) in the communication. The main aim of those actors is not to improve the communication capabilities previously seen in the Web, but to exploit and monetize the communication act. A recent definition of social media includes a reference to its cost [7]: “*Social*

media are inexpensive and easily accessible digital tools that allow anyone to create an on-line identity, publish, share and access multimedia contents intended for nonspecific audiences, build personal relationships, and engage in collective actions". Creating a basic account in many social media platforms is free. However, in order to use it, one must have access to expensive hardware devices, such as smart phones, tablets, or computers. The *inexpensive* cost mentioned in this definition assumes a contemporary context in which people already own such devices, but the acquisition of these hardware elements was not (mainly) motivated by social media.

Most of the authors agree on the key importance of social media. Also, the vast amount of data produced in social media raises tremendous research opportunities. In this document, we focus on Wikipedia [118], a well-known collaborative on-line encyclopedia. Wikipedia is not a SNS, but it is indeed a social media example, as it fulfills the fundamental features of such systems:

- **Many-to-many communication.** Everyone can join Wikipedia to be an editor, and everyone can navigate Wikipedia to consume content. The platform is knowledge-centered, in opposition to the SNSs which are mostly user-centered. That is, the planned usage for Wikipedia is not about being up to date of certain user updates, but reading some page content that, in many cases, has been written with the combined actions of several editors. This does not affect the *many-to-many* style of communication, as the actions of editors across different pages are indeed consumed by many readers.
- **Non-addressed audience.** Wikipedia's target audience is the entire Web community, and no knowledge (nor any user activity regarding content edition) is kept private nor only accessible for a reduced group of users.

2.7.1 Wikipedia

Wikipedia is a non-profit collaborative project aiming at gathering as much human knowledge as possible to make it freely accessible for everyone in the world. Individual users contribute — on a voluntary basis — bits and pieces of that knowledge to Wikipedia. People contributing to Wikipedia are called *editors*.

Wikipedia is organized in pages. Each page is an encyclopedic entry about a single entity or concept. Also, Wikipedia is organized in different language chapters, where each chapter contains pages written in a certain language. Each idea or concept has a single page within a chapter, but usually a certain concept has several entries in the whole Wikipedia, as it appears in different chapters. There is no warranty that pages in different chapters describing the same entity will contain the same information or will be free of contradiction. This is perceived as a cultural barrier, as the knowledge stored in a certain Wikipedia chapter remains inaccessible for users who cannot understand the chapter's language. In order to tackle this issue, there is an ongoing project⁵⁶ to create an abstract Wikipedia. This abstract Wikipedia would be written in an abstract language for which automatic translators to any other languages should be developed [119]. This project is still in an early development stage though.

⁵⁶https://meta.wikimedia.org/wiki/Abstract_Wikipedia Accessed in 2022/05/03.

By February 2022, the total number of pages written in all Wikipedia chapters amounts to more than 186 million⁵⁷. The biggest Wikipedia chapter in size terms is the English one, which has alone more than 42 million different pages. The number of active editors in this chapter, i.e., people who have added some content, is higher than 371.000 (just during the last year).

Wikipedia has some features regarding the quality of the content expected to be found that distinguishes it from average user-centered social media platforms.

- The aim of the platform is to produce encyclopedic-like knowledge, so the writings style is, in general, more formal.
- Each page is usually edited by several people. This not only means that several editors write content, but also that several users review and correct the content of other editors. Wikipedia is an auto-moderated community.
- Wikimedia encourages editors to add external references (i.e., references that are not other Wikipedia sources), so only trusted information is included in the platform. The usage of reliable references is recommended but not mandatory though.
- Although the actual content of a page is decided by its editors, Wikipedia pages of every chapter follow similar templates, so one can expect to find some widely accepted structures or schemata to organize the data. These structures allow human readers to easily locate key pieces of information in a Wikipedia page, but also allow automatic agents to find these pieces of knowledge by means of *web scraping*⁵⁸. Some structures frequently observed are:
 - **Abstract.** This text piece is placed before the rest of the information. It is supposed to contain few sentences which aim to be a summary of the concept described in the page.
 - **Infobox.** Infoboxes visually consist of tabular information placed at the top-right corner of a Wikipedia page. Infoboxes are composed by standalone pieces of data organized in rows. Each row contains a property and a value for that property. Infoboxes can be organized in subsections using internal headings.
 - **Index and section headings.** When the content of a page is large enough, it is frequently organized in sections (and possibly subsections). Each section is identified with a heading that is listed in a clickable index placed right after the page's abstract.

Generally speaking, the described features cause that Wikipedia's content has a higher quality compared to many other social platforms in dimensions such as orthographic and grammatic correctness, content veracity, and homogeneous structure. Needless to say, in a platform with such number of users and editors, these principles are not always successfully implemented. Wikipedia's content has issues

⁵⁷All the Wikipedia statistics offered in this section has been taken from Wikimedia's stats page. Wikimedia is the foundation owning and maintaining Wikipedia, as well as some other projects such as Wikidata, or Commons. These statistics can be queried at <https://stats.wikimedia.org/> Accessed in 2022/05/03.

⁵⁸Techniques of *data scraping* aim to automatically extract information from sources designed to be human-readable rather than machine-processable. The specific task of web scrapping consist of extracting valuable information from pieces of HTML code.

related to out-of-scope content, incorrect or biased statements, unreferenced claims, incompleteness, or different forms of vandalism [120].

The Wikipedia project have also a couple of features that makes it especially suitable for the purposes of our thesis.

- **Interlinked pages.** Wikipedia editors are encouraged to use links to other Wikipedia pages. When a certain Wikipedia page a is mentioned for the first time in another Wikipedia page b , editors are supposed to turn this raw textual mention into a hyperlink from b to a . On the one hand, this feature let processing Wikipedia as a directed graph of pages pointing to each other. On the other hand, those links generated and maintained by Wikipedia's community allow to trivially perform some Named Entity Recognition⁵⁹ tasks.
- **Relation with some KGs.** Wikipedia has a close relation with some KGs relevant in this thesis:
 - **Wikidata.** Both Wikidata and Wikipedia are projects of the Wikimedia foundation. As already mentioned, Wikipedia pages usually include infoboxes. Wikipedia allows for importing infoboxes' isolated pieces of information from Wikidata. Wikidata, unlike Wikipedia, is not separated in language chapters. Each concept and piece of data is represented a single time and potentially maintained by users of any language. Multilingualism in Wikidata is implemented by associated `rdf:langString` literals in different languages to each entity.
This relation between the two sources allows for an automatic propagation of changes in Wikidata to infoboxes in Wikipedia pages. This possibility causes that infoboxes no longer must be manually maintained by individual human editors, which implies less manual work and avoid potential contradictions between different Wikipedia chapters.
 - **DBpedia.** Same as Wikipedia, DBpedia is organized in different local chapters⁶⁰. As mentioned in section 2.1.1, DBpedia content is obtained by mining Wikipedia. In the case of the Wikipedia English chapter, every Wikipedia page has also a DBpedia page. Moreover, the URL of a Wikipedia page can be trivially mapped to its corresponding DBpedia URI.

These two features are relevant in our thesis context, especially in chapter 5, where we need 1) to find swapped knowledge between a corpus written in natural language and a RDF KG, and 2) to map entities or relations observed in a text to adequate URIs. The relation between Wikipedia and Wikidata, and specially between Wikipedia and DBpedia, can be used to that end.

2.8 Natural Language Processing

Natural Language Processing (NLP) refers to a set of computer science, linguistics, and Artificial Intelligence (AI) techniques concerned about how to process raw human language using automatic agents. The areas explored by NLP include, but are

⁵⁹The concept of Named Entity Recognition will be developed in section 2.8.

⁶⁰At the time of this writing, there are 20 different local DBpedia chapters. Information available at <https://www.dbpedia.org/members/chapter-overview/> Accessed in 2022/05/03.

not limited to, Information Retrieval (IR), automatic translation, natural language generation, voice recognition, or sentiments analysis.

Extracting machine-readable knowledge from a text corpus such as Wikipedia requires to perform some sort of NLP. There are many approaches to achieve such a goal though. Most of the NLP techniques require or are benefited from performing *pre-processing* or *text normalization* on the target content before trying to extract actual semantics. Text normalization includes techniques such as the following ones:

- **Tokenization.** “Tokenization is the task of splitting the input text into very simple units, called tokens, which generally correspond to words, numbers, and symbols” [121]. This is not a trivial process though. The simplest approach of tokenization consists in splitting some content using white spaces as delimiters between tokens. This is not enough to tokenize sequences such as ‘I’m’, which should be transformed into the pronoun ‘I’ and the verbal form ‘am’. Compound works or names with several words should also be identified as single entities. Some constructions specially observed in social media, such as the usage of emojis, hashtags, or mentions to other users, must be treated in special ways too.
- **Stemming.** Stemmers identify the *stem* of a word. For example, *swim* is the stem of the words *swimmer* and *swimming*. Basic stemming algorithms simply strip off word affixes. Stemming is especially useful for IR systems, as it brings together lexico-syntactic variants of a word which have a similar meaning [121].
- **Lemmatization.** It is the process of determining the dictionary form of a word [122]. Lemmatization is a more sophisticated form of stemming, as it tries to identify actual word roots. For instance, as already mentioned, a stemming process could map *swimmer* to the stem *swim*, as this process merely consists in detecting and removing an affix. However, being able to map the word *swum* to the lemma *swim* requires further computation.
- **Sentence splitting.** This process, also commonly referred as *sentence detection* or *sentence segmentation*, is the task of separating text into its constituent sentences. Simple approaches separate content using lists of punctuation marks such as full stops, exclamation marks or question marks. Many sentence segmenters use abbreviation lists that include sequences such as “Mr.”. These abbreviations use full stops that must not be used to split two sentences. Other issues related of sentence splitting are related to texts with special structures, such as content in addresses or bullet lists [121].

Text normalization techniques are handy for simplifying natural language and decompose it in smaller units. However, further processing is required to extract the actual meaning of a text. Let us suppose that we need to analyze the following sentence: “New movie times for *Indiana Jones and the Last Crusade* in Indianapolis, *Indiana!*”. For a human reader, it may be clear that the last mention to *Indiana* refers to a city in the state of *Indianapolis (United States)*. Also, that the first *Indiana* refers to the fictional character *Indiana Jones*, and the whole sequence *Indiana Jones and the Last Crusade* refers to a movie that seems to have new times in *Indiana’s* theaters. Such conclusions involve mental processes such as context comprehension or usage of background knowledge. Producing NLP systems that can get to similar conclusions is a complex task. Many different approaches have been proposed to achieve this goal. Those approaches include, but are not limited to, the following ones:

Regular expressions - Regular expressions are implemented in most programming languages and text processing tools. They allow developers to define text patterns (at character level). The target text is analyzed looking for matches for those patterns, so further operations can be performed over the matching sequences.

Regular expressions in NLP are mostly used to locate special sequences in the text, such as numerical pieces of data, dates, zip codes, phone numbers, URLs, passports, email addresses, or small constructions which must be mapped to some other content in normalization stages.

Regular expressions are frequently used in pre-processing tasks such as tokenization [123]. However, we can also find examples of complex systems heavily relying in this simple mechanism. *ELIZA* is a well-known example of a *chatbot*⁶¹ who tried to emulate conversations with a Rogerian psychotherapist[124]. Using regular expressions, *ELIZA* was able to recognize sequences such as “*I need X*” and reply using fixed constructions such as “*What would it mean to you if you got X?*”

Named Entities, Entity Linking, and Part of Speech - Named Entity Recognition (NER), and Part of Speech (POS) analysis are sequence labeling tasks, i.e., they receive a text input and they assign a label to some words (or every word) in the text.

NER involves the identification of proper names or Named Entities (NEs) in texts [123]. Sophisticated NER techniques not only detect NEs, but also assign a class label to each one, such as *Person*, *Event*, or *Reptile*. This is known as Named Entity Classification (NEC).

Entity Linking (EL), or Named Entity Linking (NEL), aims to assign NEs a proper identifier of a certain Knowledge Base (KB) [121]. NEL systems usually rely on well-known general-purpose graphs, such as DBpedia, to associate URIs to NEs [125].

POS is concerned with tagging words with their part of speech [121]. POS assigns linguistic roles to words, such as noun, verb, pronoun, adjective, or adverb. POS analyses helps to process both the structure and the meaning of a sentence. Current POS taggers reach an accuracy of 97% for many languages (and using different algorithms) [126]. This accuracy is comparable to the human-performance for the same task.

Some techniques used to perform sequence labeling for the tasks described are Hidden Markov Models (HMM) [127], Conditional Random Fields (CRF) [128], Convolutional Neural Networks (CNNs)⁶² [129], Recurrent Neural Networks (RNNs)[130], or Transformers [131].

N-gram Language models - *Language models* (LMs) assign probabilities to sequences of words [132].

One of the simplest LM is *n-grams*. An *n-gram* model accounts for how many times a consecutive sequence of *n* tokens is observed in a text corpus. The tokens are usually words. For example, *n-gram* models can be used to predict the last token of a sequence of *n* – 1 tokens. State-of-the-art approaches to perform complex NLP task are usually more sophisticated than *n-grams*, but this technique can be used in combination with others [132].

Some public *n-gram* corpora are available to be reused. Examples of these public models are Google Books *n-grams* [133], which contain more than 800 billion tokens of *n-grams* in eight different languages, or the Corpus of Contemporary American

⁶¹Chatbots are artificial agents which try to emulate human conversations. They are able to process human textual input and produce a response related to that input.

⁶²The concept of *Neural Network* (NN) will be developed later in this section.

English (COCA) [134], which is an English 1 billion word corpus carefully curated to contain a balanced amount of information from different sources (news, social media, fiction, etc.)⁶³.

Naive Bayes - Naive Bayes is a Machine Learning (ML) algorithm used for producing automatic classifications. IN NLP, a Bayesian classifier usually processes input texts as *bags of words*, i.e., it computes the frequency in which a word is seen in a text. Texts labeled with a class are used to train this algorithm. Then, the Naive Bayes is fed with new unlabeled samples of text. The algorithm produces a label for such samples by applying similarity measures of these new bags with the bags used for training. The classifier is used for tasks such as sentiment analysis (it classifies a text as *positive* or *negative*) [135], language identification [136], and many other types of text classification [137].

Logistic Regressions - Logistic regressions are also an ML classification technique with applications similar to the Naive Bayes' ones. Naive Bayes are generative classifiers, while logistic regressions are discriminative ones.

Informally, a generative classifier tries to learn how each target class should be like. Then, an unlabeled input i is presented to the algorithm, and it tries to guess which is the class model which is more similar to i to produce a result. In contrast, discriminative models purely try to learn how to differentiate classes. Specifically, logistic regressions try to learn a function in an n -dimensional space (n is the number of features studied) with the labeled training examples. This function divides the space in two sections, which represent two different classes. Then, an unlabeled text i is computed and translated to a single point in that n -space. The position of this point in one of the sections in which the n -space has been divided is used to classify i .

Basic logistic regressions are binary classifiers. However, several logistic regressions could be combined in *multinomial logistic regression* [138] to deal with classification problems of k classes.

Support Vector Machines - Support Vector Machines (SVM) [139] are another example of ML approach with similar application to Naive Bayes or Logistic Regressions. In NLP, SVMs are frequently trained on context of high-dimension and sparse features. They achieve a high performance on many tasks with lesser amounts of training data [123].

SVMs are also binary classifiers. SVM aim to find the optimal plane that separates two classes by maximizing the distance of this plane to the classes' closest points [140]. SVMs have been used in NLP systems for different tasks, such as email phishing detection [141], opinion mining [142], or sentiment analysis [143].

Neural Networks and Neural Language models - Neural Networks (NNs) are called *neural* after a simplified model of the human neurons' behavior described in terms of propositional logic [144]. However, modern systems are no longer built on top of those initial biological inspirations.

NNs are nets of small computing units (*neurons*). Each unit receives an input consisting of a vector of values and produces an output consisting of a single value.

⁶³The last COCA update was performed in 2020: <https://www.english-corpora.org/coca/> Accessed in 2022/05/03.

FIGURE 2.17: Partial example vectors for the words *driver*, *pilot*, and *cactus*.

	DIMENSION					
WORD	car	circuit	...	desert	thorn	water
<i>driver</i>	658	151	...	0	0	4
<i>pilot</i>	742	203	...	0	0	5
...
<i>cactus</i>	0	0	...	156	767	699

NNs combine layers of neurons to produce results. They are used to solve classification problems too. NNs are based on mathematical models similar to regressions. However, NNs are, in general, considered to be a more powerful classification algorithm compared to logistic regressions⁶⁴. Also, the usual procedure to work with logistic regressions consist of feeding the algorithm with vectors composed by n hand-picked features. One may need to evaluate different configuration features to find the logistic regression producing better predictions. In contrast, NNs are usually feed with raw words, and they are able both to learn features from the text and to decide whether a feature is useful or not to classify a text.

The computing units of a NN are often organized in layers, and layers are computed iteratively. The term *deep learning* [145] refers to NN architectures using many layers. Well known techniques based on deep learning architectures are CNN and RNN⁶⁵.

Vector semantics and embeddings - *Vector semantics* consist of a model which aims to capture the meaning of a certain element by representing it as a vector in a multi-dimensional space. The dimensions in this space are other elements. Each instance is represented w.r.t. how frequently it is found being used with those other elements. When vector semantics is used as a LM in a space of words, it is called *word embeddings* or just *embeddings*.

Word embedding algorithms are frequently applied on dense vectors, i.e., vectors which do not have as many dimensions as possible words, but a representative and meaningful set of elements (50-1000) is chosen instead. Dense vectors contain less zeros than vectors in spaces of higher dimensions. Word2vec [147] or GloVe [148] are well-known software packages to work with word embeddings.

One of the basic intuitions that can help to understand word embeddings is that elements co-occurring with similar words, i.e., words having a similar vector, may have similar meaning too. Word embeddings are used to detect relations between words. In Figure 2.17, we show a partial example vector for the words *driver*, *pilot*, and *cactus*. The dimensions of this vector are the number of times that the target word co-occurs with other words.

⁶⁴The computing units of a NN can consist of regressions. However, it cannot be strictly stated that NNs are better than logistic regressions, nor, in general, that any ML algorithm is better in absolute terms than any other. The adequacy of these algorithms is always related to an application context.

⁶⁵In this subsection, we have introduced the uses of several ML approaches as classifiers for processing natural language. Although is it out of the scope of this thesis, note that ML methods can be also used in a generative way. For example, Large Language Models (LLM) such as GPT-3, which is a pre-trained massive NN, can be used to generate human-like texts [146].

As one can see, the words *driver* and *pilot* tend to co-occur with similar words in a similar frequency. This can indicate that there is a synonymy relation between them. Word embeddings can be also used for word disambiguation tasks (detecting which sense of a polysemous word is used in a certain context) or to detect some other kinds of semantic relations, such as the one existing between *pine tree* and *pineapple*.

Note that the mentioned approaches to perform some sort of NLP tasks are not mutually exclusive. It is frequent that complex NLP systems use several of the aforementioned techniques to create a pipelines systems.

2.8.1 NLP on social media

The core algorithms to process text pieces generated in social media are not different to the ones introduced in the previous section. However, the content produced in social media has some features which are not observed with the same frequency in more formal communication contexts, such as usage on non-standard abbreviations, lack of structure, higher misspelling rates, usage of emojis, multilingualism within a paragraph/sentence, or emphatic repetitions of characters.

For this reason, text normalization on user-generated content may need to execute some extra steps. Also, when using ML algorithms, it may be necessary to use training data generated in social platforms rather than reuse generic models trained with presumably correct and standard language samples.

Content generated in platforms such as SNSs raises some extra challenges, including the following ones [149]:

- **Relevance.** When performing text mining tasks⁶⁶, not every available content is relevant. There are several reasons to consider that certain content is not relevant. For example, it could be unrelated to the topic of study, it could be duplicated or quasi-duplicated, or it could had been produced by malicious agents, such as trolls⁶⁷ and bots⁶⁸ [150].
- **Recipient Identification.** Content expressed in social media can be implicitly addressed to a certain person or entity. This is known by the sender and maybe by the receiver, but the actual receiver may not be clearly stated in the text. Errors identifying the recipient of a message can alter the semantics of the message that the sender originally intended to emit.
- **Figurative Language.** Figurative Language (FL) is a linguistic phenomenon in which there is a contradiction between the literal and non-literal meaning of an utterance [151]. Typical examples of FL are sarcasm, irony, metaphor, and satire. Figurative uses of the language seem ubiquitous in social media chats and discussion forums [152]. FL may not be expressed with any particular construction at all, i.e., it should be detected using contextual information and background knowledge about the sender and the target topic. Many approaches, mostly based on ML, have been proposed to deal with FL-related

⁶⁶The concept of data mining is developed in section 2.9.

⁶⁷The term *troll* refers to accounts that post provocative, irrelevant, or out-of-scope content to cause a negative reaction with diverse malicious intentions, such as altering public opinion, boycott, or even just for fun.

⁶⁸In social media environments, the term *bot* refers to fake accounts operated by hidden automatic agents, i.e., agents pretending to impersonate actual humans.

issues [153]. The task of detecting whether a certain content out of context is sarcastic is challenging even for human judges [154].

- **Temporal frame.** Some of the content published in social media is tightly connected to time spans related to different events (both public and private) [155]. For example, a certain user could express contradictory opinions on a matter in short time periods due to changing events. While this is not a problem for understanding the content of a specific message, it becomes a problem when trying to integrate or summarize knowledge stated in different messages.
- **Context information.** The content expressed in some social media platforms, especially in those oriented to short messages such as Twitter, usually requires a higher level of contextual information compared with more formal writing contexts. This causes classic approaches to be less suitable to analyze this kind of content [156]. Some examples of such phenomena are the following ones:
 - Expressing an opinion on something without explicitly mentioning the target of the opinion.
 - Using personal references that can be only understood by a reduced group of people.
 - Equating some concept *a* to some other concept *b* assuming that *b* has some representative features. For example, one could say that a famous person is like *Cruella de Vil*, a fictional villain of the movie *101 Dalmatians*, meaning that this person is evil.

Note that Wikipedia, the social media platform target of this study, is more affected by the problems mentioned in the discussion pages associated to each article. In contrast, the actual Wikipedia articles are expected to be expressed in a more formal and neutral style, centered in a single topic, and, in general, with enough contextual information. Such features can be achieved thanks to the knowledge-centered model and the community's commitment with the auto-moderation mechanisms of the platform.

2.9 Data mining

Data mining is defined as “a process of discovering or extracting interesting patterns, associations, changes, anomalies and significant structures from large amounts of data which is stored in multiple data sources such as file systems, databases, data warehouses or other information repositories”. [157].

Data mining is a broad concept involving many potential types of inputs and outputs. There is a constant in every data mining approach though: the input usually consists of large amounts of data which are supposed to contain valuable knowledge, but this knowledge is hard to process, locate, or generalize for humans. In contrast, the output produced by data mining processes is supposed to provide relevant and actionable information regarding some features of interest of the data analyzed. The output should be easier to understand for humans or to process by automatic agents.

A certain dataset could be analyzed by different data mining processes for different purposes. For example, a web application's log could be mined to locate Denial-of-Service (DoS) attacks⁶⁹, to produce usage statistics by client (user agents, country, etc.), or to detect which are the contents getting more engagement from users. Typical outcomes of data mining processes are entity characterizations, entity discriminations, summarizations, classifications, clusterings, models for ML algorithms, and detection of outliers, associations, or trends [158].

Data mining is a key process in many systems, and it is an active research field from different perspectives, including looking for new functions, algorithms, techniques, architectures, applications, and data domains in which to apply it [157].

There are many existing approaches to perform data mining. For example, when the input consists of raw text, methods such as the ones introduced in section 2.8 can be used. Most of the times, achieving the actual goal of a data mining task requires to implement and orchestrate several techniques. Some families of data mining approaches, both to compute the data or to produce valuable outputs, can be identified [158]:

Statistical approaches - Data mining can be based in statistical approaches, i.e., the raw input data can be mapped to statistical models to perform further analysis or to build some other product on top of the model. Correlation analyses, factor analyses, discriminant analyses, clustering, regression analyses, etc., are widely used in data mining environments [158].

The tool described in chapter 4 is an example of a statistical approach. This tool mines RDF graphs to extract RDF shapes. Each shape contains the features most frequently observed among a certain group of instances. The tool's core uses a voting system. Each instance is modeled as a *voter* that casts positive votes for features that can be observed in its immediate neighborhood. Simple statistics based on frequency and ratios allow for transforming the most voted features into actual shape constraints.

Note also that the gathered statistics in a data mining process could also be a final product. An example of such system would be a study of content distribution on a large text corpus [159].

Machine Learning - ML is widely used for classification and prediction-related goals. It is frequent to use statistical approaches combined with ML when working with large amounts of data to build representative sets of training data, to select data to execute predictions or classifications, or to perform further analysis on the outputs [160]. Several examples of data mining systems which combine statistical and ML approaches has been recently produced for diagnosis, and virus profiling during the COVID-19 global pandemic [161].

Many ML algorithms and architectures have been used in text mining environments [162], including NN [163], SVMs [164], and logistic regressions [165].

Genetic Algorithms - Genetic Algorithms (GA) [166] are mainly used to solve optimization problems and are inspired in processes of biological evolution. GAs start with an initial population of individuals. Each individual can be seen as a chain of features (conceptually similar to a chain of genes). Individuals are crossed in pairs

⁶⁹A DoS attack aims to shut down a system by flooding it with more traffic or petitions than it can handle. When DoS attacks are executed from several coordinates machines, they are called Distributed Denial-of-Service (DDoS).

to create new individuals which inherit features of both of their predecessors. The population evolves in a system driven by the following concepts:

- **Fitness function.** Each individual is evaluated with this function, which produces a fitness score for each one of them. This score indicates how good it is compared to other individuals and which are the chances that it has to be selected for reproduction.
- **Selection.** Selection stages chose fit individuals to be crossed, i.e., to act as progenitors for a new generation of individuals.
- **Crossover.** A crossover function receives two individuals (progenitors) and outputs a single individual (offspring) which is a combination of the input elements. This combination is performed with a certain degree of randomness.
- **Mutation.** New individuals may not be a strict combination of their progenitors, but they can also have random mutations.

After several iterations of creating new generations, GA systems expect to have detected individuals with optimal fitness scores, i.e., individuals that represent an optimal solution for a certain problem. GA can also be stopped w.r.t. other criteria, such as reaching a maximum number of generations, or detecting a convergence of results (the new generations do not produce better scores than their predecessors).

An example of a system based on data mining which uses genetic algorithms is presented in [167]. The authors look for optimal combinations of forest fire related variables and then apply mining on forest data to visualize wildfire susceptibility maps.

Visualizations - Summaries of target data using visual and sometimes interactive elements, such as charts, maps, or graphs, are also viewed as data mining techniques by several authors [158, 168]. In this case, it is the user of a data mining-based system who seeks for the final target or valuable knowledge produced.

An example of data mining approach oriented to visualizations is presented in [169], where outlier detection is performed by Boxplot users. An alternative example is presented in [170]. The system described produces different types of visualizations based on crime data to support decision making.

Data mining is often performed in *Big Data* environments. With Big Data we refer to different formats of large and rapidly increasing datasets [171]. How to compute such amounts of data, or even how to store and access it, are issues associated to Big Data environments [160]. Frequent solutions to this problem are based in parallel computing. Cloud-based solutions are gaining momentum to perform such computations [172]. Also, the target data could be a stream instead of a static source, so it must be processed in an iterative way using stream processing architectures [171, 173, 174].

The mining system described in chapter 4 uses a workflow in which the target RDF content is transformed into a stream of triples, so large datasets can be processed using inexpensive hardware.

2.9.1 Data mining in social media

Data produced in social media are vast, noisy, unstructured, and dynamic in nature. Mining the user generated content in social media platforms raises unique opportunities which can be beneficial for users, business, public organizations, and many other agents [175].

Data mining in social media can be performed for a wide variety of purposes. However, some of the most usual general goals can be enumerated:

Community analysis - Communities in social media can be explicit or implicit. Explicit communities are those ones that can be trivially traced by using public information, such as declared friends, followers, or subscriptions to topics. In contrast, implicit communities may be discovered only by mining data such as user interactions with each other.

Different algorithms have been proposed to perform community detection under different data scenarios [176, 177], and this research area has practical applications for a number of purposes, such as producing adequate advertising for communities [178], detection of malicious users [179], or to predict future connections [180].

Sentiment analysis and opinion mining - *Sentiment analysis* and *opinion mining* can be applied to determine the public perception on a wide variety of topics on social media. Examples of opinion mining processes are brand perception [181], user profiling [182], or political sentiment [183].

Social recommendation - *Social recommendation systems* are based both in user profiling tasks and community detection. These two types of knowledge are used to suggest appropriate content for users in a social platform according to their observed behavior and the trends observed within their communities. Such information can be used to perform product advertisement, but also to select information relevant for the user by prioritizing or filtering certain contents [184]. With this, the problem of information overflow can be alleviated.

Some studies report that social recommenders can have pernicious effects, as they produce excessive homogenization of the content without actually increasing the usefulness of the service [185]. Recommendation system may even support or ease the radicalization of individuals exposed to hateful content [186].

Influence modeling - *Influence modeling* studies how users in a network can affect the opinions, consumption habits, or general behavior of other users in the network.

There are two main types of influence phenomena studied in social networks:

- **Social influence.** Individuals in a network tend to change their habits to be like influent users.
- **Homophily.** Individuals tend to bond themselves to other individuals exhibiting similar behavior or features, and are influenced by those users.

Both types of influences are studied and identified [187] so the influence graphs in social platforms can be studied or used to achieve a number of purposes, such as promoting products [188] or trying to alter the public opinion on a matter [189]. This can be done by incentivizing influent individuals to perform some actions.

Many current marketing strategies are based on sponsoring *influencers*⁷⁰ to promote their products. Some of these campaigns are not explicitly declared as advertising, i.e., influences emit opinions on a certain product without informing their audience that they have been incentivized to promote such product. Several studies analyzing the effectiveness, scope, and ethics of influencer marketing have been performed [190–192].

Information diffusion and provenance - Detecting how information is propagated in a social network is a research field with several practical applications. The study of information diffusion patterns provides insight about mechanisms for an effective spread of information across networks [193]. Also, those patterns allow for automatically discovering rumors and fake news by tracking known patterns of information spread, especially when it is found that the source of this information consists of accounts that produce that kind of content frequently [194].

Privacy, security, and trust - Active users on social media, especially on SNSs, usually face two opposed needs. On the one hand, they would like their content could reach its target audience as effectively as possible. On the other hand, they may want that this content remains private and unreachable for anyone outside of that target audience.

The target audience could consist of a quite reduced group of people, a large list of contacts or a wider and nonspecific audience. When this last option is chosen, the intended target audience may not be the whole Internet community, even if the content posted is made public. Frequently, one may have higher interest on reaching users with some particular profiles or may even want to exclude specific individuals from the target for diverse reasons. The users' privacy expectations are frequently not fulfilled. This can happen due to having an undesired audience, but also because of the actual rights and access level that the social platform implements over their published content.

SNSs implement different mechanisms to configure access to the published content to avoid undesired interactions, but these mechanisms are far from perfect. Undesired interactions include actions such as stalking, spamming, bullying, or even scamming, phishing, and clickjacking. Most of these misconducts can constitute offenses with legal consequences.

Data mining can be used to track action patterns to discover such misconducts [195], so actions to prevent or interrupt them can be implemented.

Wikipedia's content is produced by many different users, and actions such as content edition or participation in talk pages are logged and publicly available. Then, despite being a knowledge-centered platform, it can be also an object of study w.r.t. community analysis [196], social influencing [197], or opinion analysis [198] among others. However, in this document, we will not explore the user-related dimensions of Wikipedia. Our efforts will be focused on analyzing articles' content regardless of its provenance.

⁷⁰In social media environments, the term *influencer* refers to someone that has an influence over a certain on-line audience. This influence, for example, can lead their audience to buy things or affect their opinion on particular topics.

Chapter 3

Importance metrics in RDF graphs

Note: Most of the content of this chapter has been published in a paper entitled *Approaches to measure class importance in Knowledge Graphs* [199].

3.1 Introduction

This thesis' main goal is the extraction of RDF shapes from natural language content. Existing KGs can be used as support elements in this process for different purposes, such as disambiguation, maximization of impact, or KBs to implement distant supervision [200] approaches. In such a scenario, it is highly desirable to identify important topics among the KGs used as support sources.

Although the concept of *importance* seems intuitive, there is not a canonical definition of importance when applied to contents in a dataset. In the case of important topics/nodes in a KG, the adjective *important* can refer to notions such as *well-connected*, *well-defined*, *truthful*, *complete*, *abundant* etc. Importance is not only hard to define in this context, but also hard to measure. Despite this, since detecting important elements in network structures is a recurrent problem in different scenarios, several automatic techniques to determine node importance in graph structures have been developed [2, 201, 202]. The outputs produced by these techniques are used for a wide variety of purposes, including ranking/statistical reports or graph summarization techniques.

At initial stages of the thesis, we detected that most of the potential data sources that we could use to support the process were large graphs covering many different topics. Some of those sources are general purpose KGs, such as DBpedia [203], Wikidata [26], YAGO [204], and OpenCyc [205]. Some others are natural language-centered, such as WordNet [206], or FrameNet [207]. The size and variety of such projects makes it hard to understand their schemata or to locate their most important topics. This is especially true for crowd-sourced datasets, whose knowledge proceed from collaborative efforts from a heterogeneous community.

The process of gradual discovery and understanding of the contents of a large and unfamiliar KG has been called *graph exploration* [208] or *semantic data exploration* [209]. To properly achieve such a goal, one needs to perform some sort of graph summarization. Being able to synthesize a graph's content is desirable to achieve graph exploration. First, it allows consumers to decide whether a graph is suitable for their purposes. Second, it could be used to discover the most important entities, relations, or topics of the summarized source. Those summaries can be generated either in an automatic or handcrafted ways, but the bigger, more complex, or larger it is the number of different agents maintaining a KG source, the harder it is to produce accurate handcrafted summaries. Again, this is particularly edgy

in crowd-sourced or cross-domain datasets, maintained by different people, organisms, or automatic processes.

When dealing with graph-like structures, there are many existing techniques to produce different kinds of summaries [210]. Most of them focus on *classes*, which are key elements to easily describe a graph content [211–213]. These summaries frequently consist of reduced graphs that aim to contain the key elements of the original content. To achieve such a goal, the approaches need to know which are the most important elements in the target source, and automatic rankings of importance play a key role in this subtask. Importance rankings consist of top-bottom ordered lists which sort elements w.r.t. some pre-established criteria. These lists can be used to produce the mentioned graph summaries, but they can also be used as simpler standalone summaries for certain applications.

As already stated, detecting important topics in KGs can be beneficial in several ways in the context of our thesis. Our approach to do so consist of equating the notions of class and topic. We see the concept of class as a useful abstraction that groups several individuals which share some topological features using a common label (the class URI). The group formed by those individuals can be seen as a topic. For instance, the accumulated knowledge about entities such as *France*, *Germany*, *Poland*, etc., can be expressed as knowledge about *countries*. Similarly, the subgraph formed by entities such as *Metallica* or *The Beatles* can be summarized as *band* knowledge. In both cases, the topic that encompasses the subgraph associated to the mentioned entities is also their actual ontological class.

The schema associated to a class, i.e., the set of relation types which one may expect its instances to use, is also a powerful tool for an effective knowledge exploitation of an RDF source. In general, the knowledge of RDF graphs is used via SPARQL queries. Then, knowing the expected structures associated to classes also let the users choose the adequate properties and graph schemata to write effective SPARQL queries.

Due to the exposed reasons, this chapter aims to answer the following research question:

- **RQ3:** How can we identify the most important classes of an RDF graph?

It is important to remark the difference between the concepts of *importance* and *relevance*. Relevance is much easier to define and evaluate, as it is tightly bounded to a certain purpose. For instance, search engines provide rankings of elements w.r.t. its relevance for a certain query. Recommendation systems produce lists with the most relevant items for their users. All these relevance rankings are built for a purpose, which is retrieving adequate content for a certain agent or purpose. Both search engines and recommendation systems are designed to be used by some final users. Since the final goal of such system is to produce results that meet users expectations and preferences, then those users are legitimate judges to determine whether a set of results is relevant.

In contrast, the idea of importance *per se*, decoupled of a specific purpose, can be harder to evaluate. The owners of a dataset may consider that some elements are important, while the final users may think otherwise, as their expectations for the dataset could be different from the owners' ones. Without a strict definition of *importance*, it can be a matter of discussion whose criteria should prevail over the rest of opinions.

Most of the existing techniques to automatically generate importance rankings in network structures compute different graph topological features. That is, they

aim to equate different metrics of graph centrality with the notion of importance. These metrics range from plain and simple link counts to complex network theory algorithms [201].

For the specific task of evaluating the importance of classes in RDF graphs, to the best of our knowledge, few approaches have been proposed. Some alternatives consist of a simple counting of instances, or combinations of this count with general graph theory metrics [28]. Those graph theory metrics can determine class importance, but there is no difference in how they process class nodes and instance nodes.

In order to properly answer the question **RQ3**, we have designed and developed ClassRank, a novel technique to automatically build rankings of class importance in RDF graphs. ClassRank assigns an importance score to each class consisting of the sum of its instances' PagaRank score. Thus, the importance of a class is not calculated w.r.t. the centrality of the actual class node in the target graph, but w.r.t. the accumulated importance of its instances. That is, the class node is used as a label which encompasses a set of individuals.

With ClassRank, classes that, by nature, cannot have a large number of instances, but those instances are important for the graph structure, can be high-ranked. An insightful example of such a class would be the notion of *Country* in general-purpose KGs such as DBpedia or Wikidata. Country instances usually work as bridges between knowledge of many different domains: artists are born in a country, geographical features are located at a country, administrative units are organized within countries, etc. The list of countries is limited though, so the class *Country* could not compete against any class with many instances, even if such instances are barely relevant for the graph's general structure.

A recent study [28] proposed to assume that the most important classes in RDF graphs which are exposed via SPARQL endpoints are those ones that are found to be used more frequently in SPARQL logs. This notion of importance has two peculiarities. On the one hand, it is close to the idea of relevance, as it assigns importance w.r.t. the usage of the graph. However, unless usual relevance metrics, it does not rank elements w.r.t. to a single specific purpose. Since it evaluates all the traffic seen in the logs, the purposes of every user interacting with the content via the endpoint are reflected in this notion of importance. On the other hand, unlike many other importance metrics, it totally discards information related to the actual graph topology. Each node in the graph is qualified w.r.t. data outside the graph itself.

Although this notion of importance seems reasonable, and some proposed approaches to determine node importance/relevance make use of it [214, 215], log-related information is not always available. Usually, logs can only be accessed by system administrators. The authors in [28] propose to use SPARQL logs to evaluate graph-based approaches. The assumption made is that those metrics which produce rankings more similar with the log-based rankings capture a better notion of importance.

In order to evaluate ClassRank, we performed an experiment similar to the one described by those authors. We used many state-of-the-art importance metrics to evaluate class importance in Wikidata and DBpedia. Also, we analyzed class usage in public random slices of SPARQL logs in those two sources and elaborated a ranking with the obtained information to be used as reference. Then, we analyzed how similar were the reference rankings with the ones produced by each metric under evaluation. This similarity was determined using Rank-Biased Overlap [216]. The results obtained indicate that, in general, ClassRank can produce rankings more similar to the class-usage ones than any other evaluated technique.

Despite the original aim of ClassRank was simply to provide a proper way to measure class importance in the context of our thesis, we noticed that this algorithm could be used as an standalone product in many other contexts, especially for graph summarization purposes as the ones already described. For this reason, the evaluation has not been adapted to our thesis frame, but it was designed to evaluate ClassRank in a general scenario that could be more relevant to the scientific community.

This chapter's content is organized as follows:

- In section 3.2 we make an overview of all the techniques that are evaluated in our study and provide some preliminary notions that will be useful to read the rest of the chapter.
- In section 3.3, we start describing our experimental set up: sources used, evaluation methods, and algorithm configurations. Then, we offer the results of our experimentations.
- In section 3.4, we provide an extended discussion about the experimental results. We provide different explanations to justify the algorithms' performance.
- In section 3.5, we provide a review of works related to ClassRank and the elaboration of automatic importance rankings in KGs. We make special emphasis in those techniques that has not been used during the evaluation.
- Finally, in section 3.6, we expose the conclusions of our work.

3.2 Metrics

In this section, we will work with a formal definition of a graph $G = (V, E)$, where V is a set of nodes or vertexes, and E is a set of edges linking those nodes.

All the metrics under consideration will be used to produce rankings of node importance in RDF sources. For that, they all consider certain topological aspects of the target graphs. However, we can classify those techniques in two categories depending on the type of nodes and links that they compute to produce a result.

On the one hand, we introduce approaches that are applied over graphs composed only by T-BOX statements. In the interest of brevity, we will refer to them all as *Only Terminological Techniques (OTT)*. These techniques are Degree, Betweenness, Bridging Centrality, Closeness, Harmonic Centrality, and Radiality. Theoretically, these algorithms could compute graphs containing A-BOX statements as well. However, except for Degree, they all have a computational cost that does not make them suitable to be applied over KGs as large as DBpedia or Wikidata.

On the other hand, we have algorithms that compute both A-BOX and T-Box sentences of the target graph. Instance Counting (IC), PageRank, HITS, and ClassRank fall on this category. In the interest of brevity, we will refer to them as *Also Assertion Techniques (AAT)*. Since IC and ClassRank are based on class-instance relations, they both need A-BOX statements for their computation. In opposition, PageRank and HITS can be applied over any directed network structure, so they can work as AAT or OTT. In this chapter, we evaluate both algorithms in both ways. When computing the whole target graph, we refer to them as PageRank/HITS AAT. When considering just the T-BOX subgraph, we use the nomenclature PageRank/HITS OTT.

3.2.1 Importance metrics applied over schema structures

The techniques mentioned in this section are used for general network analysis and they all aim to measure node centrality. A thorough review of these approaches and similar ones is offered in [201].

3.2.1.1 Degree

The Degree is one of the simplest measures of graph centrality. The Degree of a node e is the number of edges incident to e . We will denote the Degree of a node $e \in V$ as $D(e)$.

3.2.1.2 Betweenness

The Betweenness $B(e)$ of a node e is the ratio of shortest paths between any pair of nodes $(u, v)/e \in V \wedge e \neq u \neq v$ that pass through e compared to the total number of shortest paths. Let $\sigma(G, e)$ be the function which gives the number of shortest paths in G passing through e , and $\sigma(G)$ be the total number of shortest paths in G . Then, the Betweenness $B(e)$ can be defined as:

$$B(e) = \sum_{e \neq u \neq v} \frac{\sigma(G, e)}{\sigma(G)} \quad (3.1)$$

3.2.1.3 Bridging Centrality

A bridging path is an indirect connection between two aggregate nodes in a graph, i.e., a link of two densely connected components (e.g., a domain knowledge, an organization) via a third node known as bridging node. On the top of this concept, the Bridging Centrality $BC(e)$ of a node e assigns e a score that indicates how much e acts as a bridge mainly for the nodes in its neighborhood in G . For such a goal, it combines both local and global metrics of centrality. It is based on Betweenness and the Bridging Coefficient $B_c(e)$ of a node e , which is defined as follows:

$$B_c(e) = \frac{D(e)^{-1}}{\sum_{i \in N(e)} D(i)^{-1}} \quad (3.2)$$

where $N(e)$ is the set of nodes in the immediate neighborhood of e . With this, $BC(e)$ is defined as:

$$BC(e) = B(e) \cdot B_c(e) \quad (3.3)$$

In the context of KGs, $BC(e)$ can be used to identify useful nodes linking different information topics or knowledge domains.

3.2.1.4 Closeness and Harmonic Centrality

The Closeness $C(e)$ of a node e gives a hint about how close e is to every other node in G . The Closeness score of e consists of the average of the length of the shortest paths from e to every $v/v \in V \wedge v \neq e$. The Harmonic Centrality $HC(e)$ consists of a slight modification of Closeness, computing the harmonic mean of distances instead of the average. Let $d(u, v)$ be the function that gives the length of the shortest path between u and v . With this, $HC(e)$ can be defined as follows:

$$HC(e) = \frac{1}{\sum_{u \neq e} d(e, u)} \quad (3.4)$$

Harmonic Centrality and Closeness produce an inverse sorting of elements, i.e., the reverse rank of Closeness would be the same rank of Harmonic Centrality. The element with the highest score in Closeness is the less central one, i.e., the one whose paths to every other node happen to be the longest ones. In contrast, the element with the highest score in Harmonic centrality is the most central one. Since the rest of the techniques evaluated produce scores in which the higher is the ranking position, the more important is the element, we use Harmonic Centrality instead of Closeness during our experimentation.

3.2.1.5 Radiality

Radiality, as well as Closeness or Harmonic Centrality, aims to quantify how close is a node to all the rest in a graph. Radiality is based on the concept of Diameter of a graph $\Delta(G)$, which is the maximum distance between any pair of nodes in V . With this, the Radiality $R(e)$ of a node e can be defined as follows:

$$R(e) = \frac{1}{\sum_{u \neq e} (\Delta(G) - (\frac{1}{d(e, u)}))} \quad (3.5)$$

3.2.2 Importance metrics applied over the whole graph structure

During this subsection, we present the techniques which compute both A-BOX and T-BOX statements of the target graph used in our experiments. Instance Counting, PageRank and HITS are superficially reviewed, making emphasis mainly in those aspects that are relevant for the comprehension of this chapter. We provide relevant bibliography to deeper analysis of these algorithms in their respective sections for the interested reader. In opposition, we provide a detailed review of ClassRank, our novel proposal.

As it has already been mentioned, Degree has a computational cost that makes it suitable to be applied using A-BOX knowledge. However, we have checked empirically that the results of applying Degree AAT and IC over our target datasets is nearly identical¹, so we exclude this algorithm from the AAT subsection.

3.2.2.1 Instance counting

This importance metric is tightly linked to the RDF world and, specifically, to the class-instance relation. The more instances a class has, the more important the class is. Several public and widely used data sources offer statistics about the number of instances as a clue of class importance, such as Wikidata², or offer separate files in their dumps to manage triples about instantiation, such as DBpedia³. Instance Counting (IC) is a simple and scalable importance metric.

Typically, in RDF sources, the relation of instance-class between two elements e_c and c is expressed using the property *rdf:type* in a triple $(e_c, \text{rdf:type}, c)$. However, properties with a similar semantic to *rdf:type* can be used for the same purpose, such as *'P31 - instance of'* in Wikidata.

¹In section 3.3.2.1, we provide an extended explanation for this fact

²<https://www.wikidata.org/wiki/Wikidata:Statistics/en> Accessed in 2022/05/03.

³<https://databus.dbpedia.org/dbpedia/mappings/instance-types/2019.09.01> Accessed in 2022/05/03.

3.2.2.2 PageRank

PageRank[2] is based in a notion of importance which can be informally explained with the next statement: an element gains importance if it receives more links from other elements, if those links come from important elements, and if those elements have few outgoing links. PageRank scores are values in $[0, 1]$ with a nice statistical interpretation. The PageRank score of a node e is the probability that a random surfer, starting at a random node and jumping from node to node following links, stops at node e . PageRank was originally designed to rank the importance of pages in the World Wide Web. To model the actual behavior of an Internet user that may follow links between pages, but may write as well some new URL in his browser to move to a page non-linked from the current one, PageRank uses a parameter α . The probability d that the random surfer has of getting bored of following links and jumps to a random page is $d = 1 - \alpha$.

3.2.2.3 HITS

The HITS algorithm [202] assigns two types of scores to the nodes in a graph: hub score and authority score. A given node will have a greater authority score when it receives links from nodes with a high hub score. Also, a given node will have a greater hub score when it points to nodes with a high authority score.

As well as PageRank, HITS is an algorithm designed to rank the importance of pages in web search contexts. The hub score proposed by HITS is a similar notion to PageRank: in both cases, a node gains importance when it receives links from important nodes. However, PageRank and HITS consider a different notion of importance for those incoming links. The authority score aims for finding pages that link to many other important nodes. This notion can be especially useful for web search tasks: instead of looking for the most relevant result, it finds nodes from which you can jump to many other relevant results.

HITS is usually applied over subgraphs of a certain network containing nodes and edges relevant to a given query. For example, when applied to web search, HITS does not compute the hub and authority score for each page on the Internet. Instead, it computes the connections between those pages that are detected to be relevant for a text query. Thus, both hub and authority scores are relevant to the nature of that query, and not just to the importance w.r.t to the whole network structure.

Despite this, HITS can be applied over the whole KG, and so we do in our experiments. When applying HITS, we will use the hub score in every case, which is the one with closer semantics to the notion of node importance produced by the rest of the algorithms under evaluation.

3.2.2.4 ClassRank

ClassRank, similarly to PageRank, assigns an importance score to each element ranked, which is a value in $[0, 1]$. The ClassRank score of a class c consist of the aggregation of the PageRank scores of its instances. Let $PR(e, \alpha)$ be the PageRank of e with a damping factor of α . And let $I(c)$ be a function which returns the set of all the instances of the class c . Then, the ClassRank score $CR(c, \alpha)$ of a class c with a damping factor of α can be simply defined as follows:

$$CR(c, \alpha) = \sum_{e_c \in I(c)} PR(e_c, \alpha) \quad (3.6)$$

As well as IC, ClassRank qualifies the importance of a class w.r.t. their instances. Nevertheless, while IC purely quantifies the number of instances, ClassRank can keep a balance between the quantity and quality (aka importance) of those instances.

The ClassRank scores in $[0, 1]$ have a nice statistical interpretation. $CR(c, \alpha)$ is the probability that a random surfer such as the one described for PageRank has to land in an instance of c .

Let V_C be the subset of nodes in V that are classes. While it is always true that $\sum_{e \in V} PR(e, \alpha) = 1$, ClassRank does not have a similar property. The PageRank scores of nodes with no classes are never used, while the score of nodes with more than one class are used to increase the ClassRank score of several classes. $\sum_{c \in V_C} CR(c, \alpha) = 1$ would be true if every node $e/e \in V \wedge PR(e, \alpha) > 0$ has exactly one class. Otherwise, $\sum_{c \in V_C} CR(c, \alpha) = 1$ does not have to hold.

To determine which instances give their score to a given class, ClassRank relies on the concept of *class-pointer*. Usually, each KG uses a single property to express instance-class relations, being *rdf:type* the usual choice for such a goal. In those graphs, a straightforward pick of class-pointer is *rdf:type*. However, there are occasions in which the user may find it useful to use a more flexible notion of instance-class by picking different properties.

An example of such an arguable property could be *:occupation*. Let us consider a triple (*:sarah*, *:occupation*, *:doctor*) representing that someone called Sarah works as a doctor. It cannot be said that the essential type of *:sarah* is *:doctor* but, even with that, it can be said that Sarah *is a* doctor in informal speech. Let G be a graph describing some people's jobs, where the *rdf:type* of all the individuals is *:Person*. Then, choosing *rdf:type* as class-pointer to compute G with ClassRank will produce useless results. The only class with instances would be *:Person*. However, an execution of ClassRank using *:occupation* as class-pointer, or *rdf:type* and *:occupation* at a time, will give a distribution of importance among the different occupations (classes) stated.

Some other scenarios in which it can be interesting to choose class-pointers different from *rdf:type* are ontologies or KGs where most of the knowledge is T-BOX. Then, properties such as *rdfs:subClassOf* could be used to propagate the importance of subclasses to their parent classes.

To support those cases and similar ones, we define class-pointers as properties that are supposed to be used in triples where the object is a class. This definition does not imply that there must be a strict instance-class relation between a pair of elements linked by a class-pointer. However, that is the usual case. In this chapter, to avoid verbosity, we use the term *instance* to refer to elements that give its PageRank score to a class. ClassRank can use several class-pointers in a single execution to adapt to the user needs.

Even if ClassRank is inspired and built over PageRank scores, it is important to remark that $PR(c, \alpha) \neq CR(c, \alpha)$. While PageRank measures the importance of the URI of a class within a graph, ClassRank uses this URI as a pure label to represent the accumulated importance of a group of elements whose common feature is having the same class. Actually, as it is defined, $PR(c, \alpha)$ does not have any effect on $CR(c, \alpha)$ unless c is its own instance. Formally stated, $PR(c, \alpha)$ does not have any effect on $CR(c, \alpha)$ unless it is true that $(c, p, c) \in G \wedge p \in CP(G)$, where $CP(G)$ is the set of class-pointers of G .

ClassRank's pseudo-code is formalized in Algorithm 1. Some of the conventions used in this formalization must be clarified for a proper understanding of the pseudo-code:

- We define a graph G as a set of triples $G = \{t_1, t_2 \dots t_n\}$. A triple t is a group $t = (s_t, p_t, o_t)$ (subject, predicate, and object).
- We use the macro $f_{PR}(G, \alpha)$ to refer to the standard PageRank function. $f_{PR}(G, \alpha)$ receives a graph G and a dumping factor α as input, and it returns a vector of size n , being n the number of nodes contained in G .
- We use E to denote the set of nodes contained in G , and P to denote the set of properties used in any $t \in G$.
- We denote the set of classes to be classified with E_C , and the set of class-pointer properties with P_C .
- We use f_\emptyset to denote an empty function $f_\emptyset : \emptyset \rightarrow \wp(E_C)$, i.e., a function whose domain is the empty set \emptyset and whose co-domain consist of all possible subsets (powerset) of E_C .
- In line 16, we initialize a vector of maps, and to represent each map we are using function notation. Given a certain function f , we denote its domain with $\mathcal{D}(f)$, and its graph with $\mathcal{G}(f)$. We modify the definition of a function f by adding or modifying elements in $\mathcal{G}(f)$, i.e., to define $f(a) = b$, we will use $\mathcal{G}(f)[a] \leftarrow b$.

Algorithm 1 receives the following inputs:

1. A target graph G .
2. A set P_C of class-pointers.
3. A damping factor α used for the PageRank execution.
4. A security threshold θ .
5. A set of target classes E_C , which can be empty if the target classes are not known a priori.

The threshold θ is used to ignore classes with few instances. This is especially useful when the classes to rank are not known a priori and the user of ClassRank wants to discover those classes using the class-pointers, but he prefers to discard those elements with few instances.

The algorithm returns three results:

1. The standard PageRank vector for every entity $\{e/e \in E\}$, denoted as L .
2. The ClassRank vector for every class $\{e_C/e_C \in E_C\}$, denoted as L' .
3. A matrix containing information about which entities point to which classes using which class-pointer, denoted as S

L' provides the importance of each class. S Allows to analyze the source of the importance for each class. S is useful for ClassRank executions using more than one class-pointer, as it allows the user to analyze the source of the importance for each class, i.e., which are the class-pointers that cause a bigger effect on a class score.

ClassRank can be seen as a succession of four stages:

- Preliminary stage: initializations.

Algorithm 1 ClassRank pseudo-code**Input:** $G =$ Target Graph**Input:** $P_c =$ Set of properties identified as class-pointers**Input:** $\alpha =$ Damping factor**Input:** $\theta =$ Security threshold**Input:** $E_C =$ Target classes (it can be an empty set)1: $I_c \leftarrow \emptyset$

$$2: Q \leftarrow \begin{matrix} & o_1 & o_2 & \cdots & o_m \\ p_1 & \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \\ p_2 & \\ \vdots & \\ p_n & \end{matrix}$$

▷ Stage 1

3: $L \leftarrow f_{PR}(G, \alpha)$

▷ Stage 2

4: **for each** $t_i = (s_{t_i}, p_{t_i}, o_{t_i}) \in G$ **do**5: **if** $p_{t_i} \in P_c$ **then**6: $Q_{p_{t_i}, o_{t_i}} \leftarrow Q_{p_{t_i}, o_{t_i}} + 1$ 7: **if** $E_C \neq \emptyset$ **then**8: $I_c \leftarrow E_C$ 9: **else**10: **for each** $j \in [1, |E|]$ **do**11: **for each** $i \in [1, |P_c|]$ **do**12: **if** $Q_{p_i, o_j} > \theta$ **then**13: $I_c \leftarrow I_c \cup \{o_j\}$ 14: **break**

▷ Stage 3

$$15: L' \leftarrow \begin{pmatrix} e_{c_1} & e_{c_2} & \cdots & e_{c_n} \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

$$16: S \leftarrow \begin{pmatrix} f_{\emptyset} & f_{\emptyset} & \cdots & f_{\emptyset} \end{pmatrix}$$

17: **for each** $t_i = (s_{t_i}, p_{t_i}, o_{t_i}) \in G$ **do**18: **if** $p_{t_i} \in P_c \wedge o_{t_i} \in I_c \wedge Q_{p_{t_i}, o_{t_i}} \geq \theta$ **then**19: **if** $p_{t_i} \notin \mathcal{D}(S_{o_{t_i}})$ **then**20: $\mathcal{G}(S_{o_{t_i}})[p_{t_i}] \leftarrow \emptyset$ 21: **if** $s_{t_i} \notin \bigcup_{a \in \mathcal{D}(S_{o_{t_i}})} S_{o_{t_i}}(a)$ **then**22: $L'_{o_{t_i}} \leftarrow L'_{o_{t_i}} + L_{s_{t_i}}$ 23: $\mathcal{G}(S_{o_{t_i}})[p_{t_i}] \leftarrow \mathcal{G}(S_{o_{t_i}})[p_{t_i}] \cup \{s_{t_i}\}$ **Output:** $L =$ PageRank score of each entity**Output:** $S =$ instantiation vector**Output:** $L' =$ Aggregated PageRank score of each class

- Stage 1: PageRank.
- Stage 2: Class-pointer matrix and class detection.
- Stage 3: ClassRank scores.

These stages are labeled with comments in Algorithm 1. In the following paragraphs, we make an overview of each stage.

Preliminary stage: Initializations - At this stage, we initialize some data structures that will be used during the calculations of the ClassRank scores.

I_c is a set that will contain identifiers of the ranked classes. Q is a matrix of $(m \cdot n)$, where $m = |E|$ and $n = |P_C|$. In Q , we annotate how many times a given object o_j is linked with a given class-pointer p_i .

Stage 1: PageRank - At this stage, we calculate the internal relevance of each entity in G and we store it in vector L . The computation of PageRank is a widely studied problem [217, 218]. Also, there are standard libraries for many widely used programming languages to compute it. Our implementation of ClassRank is written in Python and the PageRank computation is based on the *networkx library* [219]. Nevertheless, to cope with huge graphs, we have implemented a memory-optimized version of networkx's PageRank just for non-weighted computations⁴. This implementation could be necessary to reproduce the experiments proposed in this paper.

Stage 2: Class-pointer matrix and class detection - This stage could be divided into two phases.

In *lines 4-6*, the matrix Q is filled with values. This matrix contains the number of times that each class-pointer is used to link each node of the graph.

In *lines 9-14*, we perform class discovery if needed. When $E_C \neq \emptyset$, it means that the set of classes to be ranked is known a priori, so there is no need to execute lines from 10-14. Otherwise, the algorithm looks for nodes that are pointed at least θ times by at least a class-pointer. The set I_c is filled with the nodes fitting that condition. Those nodes are considered classes by the algorithm.

The security threshold θ has been introduced to filter wrong identifications of classes causing noise. This is especially handy in sources maintained by many agents making small editions, where human actions can cause marginal mistakes. This threshold should be used carefully, since it may also cause a certain number of false negatives for all those actual classes that are pointed less than θ times by a class-pointer.

Stage 3: ClassRank scores - The ClassRank score of each class is calculated as the aggregation of the PageRank scores of its instances in *lines 15 to 23*. This stage can be informally summarized with the next statement: if there is a high enough number of triples that have the same class-pointer as predicate and the same class URI as object, then the PageRank scores of the subjects of those triples are added to the ClassRank score of the class URI.

In *lines 17-18*, for each triple $t_i = (s_{t_i}, p_{t_i}, o_{t_i})$, we check whether o_{t_i} is a class and p_{t_i} is a class-pointer linked to o_{t_i} at least θ times. If this is true, we perform three actions:

⁴<https://github.com/DaniFdezAlvarez/classrank/blob/develop/core/external/pagerank/wespageranker.py> Accessed in 2022/05/03.

- In *lines 19-20*, we include p_{t_i} as class-pointer of o_{t_i} in the vector of maps S , only if it had not been already included.
- In *lines 21-22*, we add the PageRank score of s_{t_i} to the ClassRank score of o_{t_i} , only if it had not been already added.
- In *line 23*, we specify in the vector of maps S that s_{t_i} is instance of o_{t_i} due to the class-pointer p_{t_i} .

There is a public implementation of ClassRank available in a GitHub repository⁵.

3.2.3 Adapted importance metrics

The authors in [28] propose an approach to adapt the OTT metrics described in section 3.2.1. This adaptation incorporates A-BOX knowledge to compute the scores, so the approaches become AAT. Let T_i be a given importance metric. To compute an adaptation T'_i of T_i , the authors first propose a score normalization $N(T_i(e))$ for a given node e in a scale $[0,1]$, defined as follows:

$$N(T_i(e)) = \frac{T_i(e) - \min(T_i, G)}{\max(T_i, G) - \min(T_i, G)} \quad (3.7)$$

Where $\min(T_i, G)$ is the value of the less important node in G according to T_i , and $\max(T_i, G)$ is the value of the most important node.

The adapted metric T'_i is computed as follows:

$$T'_i(e) = N(T_i(e)) + N(IC(e)) \quad (3.8)$$

Where $IC(e)$ is the number of instances of class e . This adapted metric is an equally weighted addition of the normalized scores of IC and T_i .

We have adapted all the OTT techniques mentioned in section 3.2.1 according to this formula. Besides, we have experimented with adapted versions of the rest of the AAT techniques which are compatible with this proposal. We have combined the IC scores with PageRank OTT/AAT, HITS OTT/AAT, and ClassRank.

3.3 Experiments

We used some representative samples of Wikidata and DBpedia SPARQL logs to determine which are the classes that appear most times in the logs of these two sources. Then, we used the rankings obtained to evaluate the metrics described in section 3.2. This section describes our experiments. It is structured as follows:

- **Methodology.** We describe our evaluation methods: we explain how to measure class usage, how to build the reference rankings, and the method to compare rankings.
- **Sources.** We describe the aspects of Wikidata and DBpedia which are relevant for our evaluation. This includes both the SPARQL sample logs used to build our reference rankings and the KGs themselves.
- **Results.** We provide the results of our experiments, highlighting the most relevant facts.

⁵<https://github.com/DaniFdezAlvarez/classrank> Accessed in 2022/05/03.

3.3.1 Methodology

As suggested in [28], we have considered mentions class mentions in SPARQL queries as a reliable metric of how important a class is. Note that this metric does not execute the SPARQL queries in the logs. A class is considered to be mentioned when it appears in the query itself. We count a class mention in a query when one of the following events occur:

- The URI of the class is mentioned.
- The URI of an instance of the class is mentioned.
- The URI of an element e is mentioned, in case e is used in a triple with a property whose domain/range forces e to be an instance of a class. Let p_I be a property which links an instance to its class; let $d(p)$ and $r(p)$ be the domain and the range of the property p ; and let G be the KG under analysis. Then, formally, a class c is considered to be mentioned in a query if e is mentioned and it is true that $((e, p_a, o) \in G \wedge c \in d(p_a)) \vee ((s, p_b, e) \in G \wedge c \in r(p_b))$, even if $(e, p_I, c) \notin G$.

The importance rankings are built so the classes are sorted w.r.t. its total number of mentions. The classes with more mentions appear at the top of the list.

3.3.1.1 Reference rankings: human-generated traffic vs machine-generated traffic

We produce two different lists for each studied source. One considers every entry available in the logs. The other one computes just those entries associated with human-performed queries. With requests made by humans we mean requests caused by people writing and executing ad-hoc SPARQL queries or performing small tasks in some applications that trigger a single/few queries. Usually, machines generate much more traffic than humans. To provide some revealing numbers, the sum of requests performed by the top-10 most active IPs in our DBpedia's log, which is associated with machine agents, represents 44% of the total requests. If we consider the top-100 IPs, this percentage grows to 77%.

Human traffic is not a more reliable notion of importance than machine traffic, nor vice-versa. However, it is relevant to check the difference between human-generated queries and machine generated traffic, due to the notorious impact that very few machine agents usually have over the whole log. The notion of importance purely based on human actions seems to adopt a more general point of view, not so polarized by automatic voracious consumers of the endpoint. Distinguishing between requests performed by humans and requests performed by bots or applications has already been purposed in some other studies of SPARQL logs [220].

To avoid verbosity, in this chapter, we will use the following abbreviations:

- **Human Hosts (HH)** to denote log entries associated with humans.
- **Machine Hosts (MH)** to denote log entries associated with machines.
- **Every Host (EH)** to denote all log entries.

3.3.1.2 How to compare the rankings

The main goal of our experiments is to compare the rankings produced by class-usage in SPARQL logs with the different lists produced by the techniques under evaluation. These two rankings have two peculiarities. First, it is feasible to have tied elements. Second, the significance of changes in the top of the ranking is higher than changes in the low spots. Search engine results or classification in sports are insightful examples of that last feature. When comparing two search engines, the first results shown to the user are much more relevant than the ones in position 100th. Similarly, the event of a player climbing from the second seed to the top seed in a given sport receives more social attention than jumps in deeper regions of the ranking. Importance rankings in RDF sources are used to prioritize some elements for different tasks or to get a general idea about the content of a given source. Then, the top-ranked elements are more relevant than the low-ranked ones.

It is desirable to use a metric that can compare the similarity of two rankings naturally handling these two features. We have found that Rank-Biased Overlap (RBO) [216] meets our requirements. Originally, RBO is defined as a distance measure between two rankings, where 0 means minimum distance and 1 means maximum distance. However, it can be trivially transformed into a metric by calculating $1 - RBO$, where 1 means maximum similarity. From this point, we will work with the definition of RBO as a metric.

RBO checks the overlap of two rankings at incrementally increasing depths. The elements checked at each depth d are those in rank $[1, 2, \dots, d]$. Since the first element is checked looking for overlap at every iteration, this element has the greatest impact on the results. The following element with more importance over the score will be the second one, and so on. At each iteration, RBO computes the ratio of overlapped elements. It produces a result by adding all those ratios weighted using an infinite series of weights whose sum converges always to a fixed value. The weights can be configured to give a certain amount of importance to a region of the top rank using a parameter p .

The p parameter has a nice statistical interpretation. It models the user's persistence when performing a manual checking of the rankings. Low values of p arbitrarily decrease the probability that a user has to keep exploring ranks, and vice-versa. The extreme case $p = 0$ causes that the only position checked is the first one. With $p = 0$, RBO gives a result of 0 (no overlap) when the first element of both rankings is not the same, or 1 (perfect overlap) otherwise. The rest of the ranking would be ignored. The higher is the value of p , the less probable it is that the user stops exploring the ranking.

Greater values of p arbitrarily increase the importance of wider prefixes of the rankings. Each iteration k will always have a greater impact over the results than $k + 1$, but greater values of p decrease that difference. p can also be interpreted as a parameter to configure the exact amount of importance over the final score that a given prefix length has. For instance, a value of $p \simeq 0.9$ gives an importance of 86% to the top 10 elements. This means that the sum of weights of the first 10 iterations of RBO will be 0.86. Although there is not a function to obtain a value of p for a couple of chosen values of importance and prefix length, the authors in [216] provide the following useful equation:

$$W_{RBO}(1 : d) = 1 - p^{d-1} + \frac{1-p}{p} \cdot d \cdot \left(\ln \frac{1}{1-p} - \sum_{i=1}^{d-1} \frac{p^i}{i} \right) \quad (3.9)$$

In Equation 3.9, d is the depth or prefix length, and $W_{RBO}(1 : d)$ is the accumulated weight of a ranking in positions 1 to d .

We have developed a script $f_p(d, w, \theta_e)$ which receives a length d , a weight w , and an error threshold θ_e , and it returns a value p which approximately solves the equation 3.9 for d and $w = W_{RBO}(1 : d)$. The script computes Equation 3.9 for d with different values of p , obtaining each time a result w_{p_i} . The script stops when it finds a $p_i / |w_{p_i} - w| < \theta_e$, and returns p_i . This script allows us to find accurate enough values of p for any chosen pair of prefix length and accumulated weight.

The sum of the weights at each depth of RBO always converges to a fixed value. This makes RBO an adequate candidate to compare infinite rankings without having the infinite tail's importance dominating the finite head. When computing RBO for a given depth, even if this depth is the size of the compared rankings, the algorithm produces two results: rbo_{min} and rbo_{res} . The value rbo_{min} is the overlapped score obtained after having checked the target rankings until depth d . rbo_{res} is the residual score that would have been added to the result in case the explored rankings had infinite but equal and equally sorted elements beyond depth d . With this, we can have that the max possible score for infinite lists is $rbo_{max} = rbo_{min} + rbo_{res}$.

Then, RBO can be defined as a function $f_{RBO}(R, L, p, d) \rightarrow rbo_{min}, rbo_{res}$. It compares two rankings R and L until depth d , with a user persistence modeled by p , and returns a score rbo_{min} in the range $[0, 1]$, and a residual rbo_{res} based on the assumption that R and L can have infinite elements.

The authors in [216] provide a formula to express RBO as a single point rbo_{ext} instead of a range. This formula extrapolates the tendency observed until depth d and assumes that it will stay stable along the infinite tail, and it produces a score rbo_{ext} where $rbo_{min} \leq rbo_{ext} \leq rbo_{max}$. In our experiments, we will use rbo_{ext} to obtain a single score point of similarity between two rankings.

The rankings compared will always have the same number of elements, but the way in which ties are represented may cause that they do not have the same number of ranks. When two elements have an identical score within the same ranking, they are both assigned to rank k , and the element after them is placed at rank $k + 1$. This means that the total number of ranks could be smaller than the total number of elements. In our experiments, we will always execute RBO with the longest possible depth, i.e., the depth of the ranking with more ranks.

A deeper discussion about the convenience of this technique in scenarios like ours, as opposed to classic approaches such as Spearman [221], in which all the elements have the same impact over the final score, is provided in [216].

3.3.2 Sources

3.3.2.1 DBpedia

Knowledge Graph - We used the English chapter of DBpedia in our experiments. PageRank and ClassRank compute links between entity or class nodes, i.e., the results are unaffected by triples whose object is a literal or a blank node. The contents exposed in DBpedia's SPARQL endpoint are publicly available as text dumps. Different aspects of the graph are serialized in different files. The DBpedia subgraph used for our experiments consist of the following files:

- Mapping-based objects.
- Infobox properties.
- Instance types.

Log size	58.771GB
N° of entries in the log	74,281,130
N° of MH entries	74,187,809
N° of HH entries	93,321
N° of total class mentions	80,562,206
N° of direct class mentions	42,788,969
N° of instance mentions	33,609,979
N° of class mentions inferred by domain/range	4,163,258

TABLE 3.1: Statistics about the DBpedia SPARQL logs used

- Infobox properties mapped.
- Person data.
- Specific mapping based properties.
- Topical concepts.

All the links to download these files are available on-line⁶. Such links point to the in-force version of DBpedia content at the moment in which the SPARQL logs were generated.

The classes ranked are the ones in the DBpedia ontology⁷. Also, the subgraph used to compute the OTT techniques has been the DBpedia ontology itself.

Logs - The DBpedia logs used for this experiment contain queries made to the DBpedia SPARQL endpoint in fourteen different random days during 2017. The log entries are split into fourteen files. Each file includes every SPARQL request against the endpoint within a single day. The main features of the log files are provided in Table 3.1.

We counted class mentions using the criteria described at the beginning of this section. The scripts used to perform such a mining task are publicly available in a GitHub repository⁸.

Each line in the logs contains data related to a single request to the endpoint. The version of the logs that we were able to compute was filtered and anonymized to preserve users' privacy. These anonymized logs have been provided by OpenLink Software and are available for use⁹ under a CC-BY license (credits to OpenLink)¹⁰

We could use the following information for each entry:

- Hashed IP from where the request was performed.
- HTTP request. SPARQL queries are embedded in GET requests.

⁶<https://github.com/DaniFdezAlvarez/classrank/tree/develop/experimentation/doc/dbpedia> Accessed in 2022/05/03.

⁷The version of the DBpedia ontology used in this experiment is available in the following link: <https://github.com/DaniFdezAlvarez/classrank/blob/develop/experimentation/doc/dbpedia/dbo.ttl> Accessed in 2022/05/03.

⁸https://github.com/DaniFdezAlvarez/classrank/tree/master/experimentation/query_mining Accessed in 2022/05/03.

⁹Log description and download link: <https://github.com/DaniFdezAlvarez/classrank/tree/develop/experimentation/doc/dbpedia#logs> Accessed in 2022/05/03.

¹⁰License description: <http://data.weso.es/classrank/logs/COPYING.txt> Accessed in 2022/05/03.

- Timestamp of the request truncated to hour precision.
- HTTP status code (200 OK, 404 Not found, 5XX server error, etc.).

When mining logs, there is not a perfect technique to distinguish between human and machine search sessions, or even to properly perform the task of identifying a single search session. Also, the most accurate approaches use to rely on information that is not available in the version of the logs that we have been able to analyze, such as user agent, more precise timestamps, or user identifiers. For instance, authors in [220] distinguish between queries performed by humans, which they call *organic*, and queries performed by automatic processes, which they call *robotic* queries. The SPARQL logs that they use do not contain IPs, but they do include user-agents. They use this field to detect browser user-agents, which are usually connected to organic queries, and some other agents linked to known applications used by humans.

In our scenario, we combined the information of hashed IP and timestamp to detect hosts that have a human-like amount and rate of requests. We picked an arbitrarily low number of requests by hour within a single day. Any IP showing a request rate under that threshold was classified as belonging to organic agents. The chosen threshold has been 2.

We are aware that there are several situations in which this heuristic and this arbitrary threshold may cause false positives and false negatives, such as the following ones:

- A true human agent produces too many requests within a single day, which discards not just the requests of that day but also every log entry related to the same IP.
- A machine agent produces a low enough number of requests every day.
- An IP is linked to different routers on different days due to, for instance, Dynamic Host Configuration Protocol (DHCP) changes.

The mentioned issues are hard to prevent without more precise information about each log entry. However, the threshold has been chosen to pick IPs with high chances of belonging to human agents by sacrificing recall yet having a representative sample of human-related entries.

OTT metrics - In this subsection, we describe how we computed techniques over a subgraph containing just T-BOX statements. This subgraph is the version of the DBpedia ontology in force at the time in which the logs were generated.

The authors in [28] use Degree, Betweenness, Bridging Centrality, Harmonic Centrality, and Radiality to measure class importance by applying them to graphs containing only T-BOX statements. In our paper, we used the same approach for those techniques. Except for Degree, whose complexity is linear to the number of nodes, all the aforementioned techniques need the computation of the shortest paths between all the nodes in the graph. This requires at least a computation time of $O(V \cdot (V + E))$ [28], being V the number of nodes and E the number of edges. Thus, these algorithms are hard to compute in a source such as the analyzed section of the English chapter of DBpedia, with more than 110M triples.

Also, we have applied HITS and PageRank over the DBpedia ontology. The damping factor for the PageRank execution was set to $\alpha = 0.85$, which is standard and most usual configuration of PageRank [222].

AAT metrics - We have applied HITS, PageRank, IC, and ClassRank over the whole KG.

Even if Degree's complexity is low enough to execute this technique as AAT, we have not included its computation in this chapter because of its similitude with IC. While IC only considers links class-instance, Degree computes every incoming or outgoing link to rank a class. However, for the top positions of our rankings, the vast majority of those links are the instance-class relations accounted by IC, which leads to nearly identical results for IC and Degree AAT.

To build the ranking of classes of PageRank, we filtered all the A-BOX elements in the obtained PageRank vector and sorted the remaining T-BOX terms in decreasing order w.r.t. to its score. The damping factor for the PageRank execution was set to the standard value $\alpha = 0.85$.

The ClassRank scores are built on top of the PageRank ones described in the previous subsection, so the setting $\alpha = 0.85$ was used. When executing ClassRank, the set of target classes is known a priori. Therefore, there was no need to perform class discovery in stage 2 of Algorithm 1. The property *rdf:type* was the only class-pointer considered. Since the only property linking an A-BOX term with any element in the DBpedia ontology is *rdf:type*, this is a straightforward decision in the context of our experiment. The same decision was taken to compute IC, i.e., the only property that we considered to link an instance to its class is *rdf:type*. Also, since the set of classes to classify is known a priori, the value that makes sense for ClassRank's security threshold is $\theta = 0$. We do not want to discard any class of the DBpedia ontology from the results.

In section 3.2.3, we described an adaptation of OTT approaches with IC scores. We have applied this adaptation to all the techniques mentioned in section 3.3.2.1, but we have also computed an adaptation of PageRank, HITS, and ClassRank over the KG to combine their scores with IC scores.

Reference rankings - In Table 3.2, we include the top-20 elements of each reference list. The whole lists are publicly available¹¹.

3.3.2.2 Wikidata

Knowledge Graph - Wikidata, as DBpedia, is a well known general-purpose LD source. However, these two projects have some crucial differences that affect our experiments.

Wikidata models two types of elements within its KG: entities and properties. Entities are represented with an ID starting by Q and followed by an integer (ex: Q5 stands for *human*; Q6256 stands for *country*). Properties are identified with a 'P' and an integer (ex: P31 stands for *instance of*; P279 stands for *subclass of*).

Within the entities, we can conceptually make a distinction between classes (such as 'Q6256 - *country*') and instances (such as 'Q30 - *United States of America*'). However, there is no distinction in the way these two elements are managed within Wikidata's environment. Both classes and instances are maintained by community editions and Wikidata does not provide a list of classes. It categorizes as a class any element in the KG that is an object in a triple with the property 'P31 - *instance of*' or a subject/object in a triple with 'P279 - *subclass of*'¹².

¹¹<https://github.com/DaniFdezAlvarez/classrank/tree/master/experimentation/doc/dbpedia#user-content-mining-logs> Accessed in 2022/05/03.

¹²https://www.wikidata.org/wiki/Wikidata:WikiProject_Ontology/Classes Accessed in 2022/05/03.

Pos.	Human Hosts		Every Host	
	Class	Mentions	Class	Mentions
1	dbo:Album	1,342	dbo:Place	16,567,840
2	dbo:Company	667	dbo:Airport	16,082,487
3	dbo:Place	603	dbo:CareerStation	4,264,500
4	dbo:Airport	568	dbo:Band	4,065,636
5	dbo:Person	551	dbo:Person	3,228,266
6	dbo:Country	354	dbo:MusicalArtist	2,703,628
7	dbo:SoccerPlayer	319	dbo:Organisation	2,240,076
8	dbo:Settlement	301	dbo:PopulatedPlace	1,958,866
9	dbo:City	225	dbo:Company	1,934,215
10	dbo:RadioStation	178	dbo:Language	1,851,015
11	dbo:Film	166	dbo:Artwork	1,780,684
12	dbo:Writer	146	dbo:Device	1,774,061
13	dbo:OfficeHolder	145	dbo:Settlement	1,442,655
14	dbo:MilitaryConflict	129	dbo:VideoGame	1,126,766
15	dbo:Software	117	dbo:Album	1,014,906
16	dbo:MusicalArtist	114	dbo:Film	957,870
17	dbo:IceHockeyPlayer	113	dbo:OfficeHolder	841,060
18	dbo:VideoGame	108	dbo:City	834,053
19	dbo:Drug	108	dbo:SpTeamMember	828,302
20	dbo:Scientist	91	dbo:Writer	774,113

TABLE 3.2: Top20 elements for HH and EH entries in DBpedia

With this criterion, we have found 2,477,094 classes. The subgraph of elements linking those elements contains 13,791,207 triples. In opposition to the DBpedia’s case, this T-BOX subgraph is too big to apply some of the techniques mentioned in section 3.2.1. Specifically, Betweenness, Radiality, Harmonic Centrality, and Bridging Centrality would require huge computational power and execution time.

Wikidata barely uses properties of external ontologies. With few exceptions such as *rdfs:label*, which is used to link ‘Q’ items with their denominations in different languages, the properties used to express most of the relations are defined within its own ontology and referenced with a ‘P’ ID. The class-instance relation is not expressed with *rdf:type*, but the equivalent Wikidata property ‘P31 - instance of’.

Logs - Wikidata made public some SPARQL logs with random samplings of valid queries in different time frames¹³. All these logs have been anonymized to preserve users’ privacy. Each log entry contains the original SPARQL query with variable names and most of the literals substituted by generic placeholders. There is also some metadata associated with each query. Wikidata classifies each entry as either ‘robotic’ or ‘organic’. A query is labeled as robotic when its user agent is not a web browser or when there is a non-human rate of queries coming from the same IP in a time span. Otherwise, the query is considered organic.

This label let us build the HH and EH rankings without further computations of the rest of the metadata. We used organic queries to elaborate the HH ranking and both organic and robotic queries for the EH ranking.

¹³https://iccl.inf.tu-dresden.de/web/Wikidata_SPARQL_Logs/en Accessed in 2022/05/03.

Log size	50,358 GB
N° of entries in the log	24,417,813
N° of MH entries	23,545,258
N° of HH entries	872,555
N° of total class mentions	105,939,034
N° of direct class mentions	22,094,776
N° of instance mentions	83,844,258

TABLE 3.3: Statistics about the Wikidata SPARQL logs used

The logs used in this experiments are publicly available¹⁴. They contain a random sample of queries performed between 2018-02-26 and 2018-03-25. A summary of the log's content is offered in Table 3.3.

In Wikidata, the process for creating a new property requires a thorougher community moderation than the process for creating a new 'Q' item. Even if any Wikidata user can propose a new property, this proposal needs to be discussed and accepted by the community. At the moment in which our experiments were performed, a total of 8,882 properties were defined in Wikidata¹⁵. However, some of those properties are not used any longer or have been removed.

Not all the properties include domain and range restrictions in their definition. Also, some of these restrictions aim to be pure information for Wikidata users instead of actual ontological restrictions that may invalidate some triples when used wrong. Properties usually define domain and range restrictions as choice lists. For instance, '*P106 - occupation*' is defined to be used in triples whose subject should be an instance of one class in a list of 8 elements, which includes '*Q5 - human*', '*Q729 - animal*', or '*Q21070598 - narrative entity*'.

Those domain and range definitions in some cases, and the absence of constraints in some others, do not make it possible to count class mentions in the logs via domain or range inferences. Then, when mining the Wikidata logs, we only count as class mentions 1) mentions to the class URI itself, and 2) mentions of its instances' URIs.

OTT metrics - The high number of classes on Wikidata and the size of its T-BOX subgraph discard the computation of too complex techniques such as Betweenness, Radiality, Harmonic Centrality, and Bridging Centrality.

The set of techniques applied over the T-BOX subgraph of Wikidata consist of Degree, HITS, and PageRank. As usual, PageRank damping factor was set to $\alpha = 0.85$

AAT metrics - We computed the whole KG with HITS, PageRank, IC, and ClassRank. Also, we have computed the adaptation with IC scores of HITS, PageRank, and ClassRank. As usual, the PageRank damping factor was set to $\alpha = 0.85$.

We have computed several settings of ClassRank's class-pointers P_C against Wikidata. Since '*P31 - instance of*' is equivalent to *rdf:type*, the most straight-forward setting is using $P_C = \{ 'P31 - instance of' \}$, so classes are scored w.r.t. pure instance-class relations. However, in opposition to the DBpedia case, there are many different

¹⁴https://analytics.wikimedia.org/datasets/one-off/wikidata/sparql_query_logs/2018-02-26_2018-03-25/2018-02-26_2018-03-25_all.tsv.gz Accessed in 2022/05/03.

¹⁵The current number of declared properties (some of them possibly already removed) is 9,973 Accessed in 2022/05/03.

P31- instance of	P8006- footedness	P4680- constraint scope
P279- subclass of	P4882- segmental innervation	P1354- shown with features
P1750- name day	P6884- target muscle	P548- version type
P1622- driving side	P1913- gene dupl. assoc. with	P4390- mapping rel. type
P8030- size designation	P4954- may prevent	P1914- gene ins. assoc. with
P3358- positive prog. pred.	P922- magnetic ordering	P2894- day of week
P8225- is metaclass for	P2352- applies to taxon	P2597- Gram staining
P6437- day of regular release	P660- EC enzyme classif.	P21- sex or gender
P5102- nature of statement	P1480- sourcing circumstances	P6216- copyright status
P6106- uses capitalization for	P2443- stage reached	P556- crystal system
P873- phase point	P1310- statement disputed by	P4794- season starts
P2308- class	P4649- id. of subj. in context	P6224- level of description
P1910- decreased expression in	P1642- acquisition transaction	P91- sexual orientation
P3357- negative diag. pred.	P4850- perm. food additive	P8127- tournament format
P2577- admissible rule in	P1917- posttranslat. mod.	P4224- category contains
P6118- season ends	P8431- course	P3150- birthday
P8115- eligible recipient	P1915- gene inv. assoc. with	P404- game mode
P970- neurological function	P1918- altered reg. leads to	P105- taxon rank
P3294- encoding	P5439- research measurement	P7937- form of creat. work

TABLE 3.4: Properties with a class ratio ≥ 0.99

properties linking instances and classes in Wikidata, which let use different P_C settings.

A class-pointer could be any property defined to be used in triples where the object is a class. However, properties in Wikidata are not defined strongly enough to find class-pointer candidates by checking their constraint definitions. Also, some properties that could be automatically detected as class-pointer candidates are not used as so in the KG. For example, ‘*P413 - position played on team*’, which is supposed to point to classes standing for special roles in different sports, is just used to point to actual classes 71.22% of the times.

Then, to obtain a list of class-pointer candidates, we computed the actual usage of every property p in the whole KG to obtain a ratio $r_p = \frac{U_{c_p}}{U_p}$, where U_p is the number of times that p is used in any triple, and U_{c_p} the number of times in which p points to a class. Then, we sort the properties in descending order w.r.t. r_p . The list of properties and its associated ratio is available on-line¹⁶. We tested every combination of properties above a certain ratio, starting at 1.0 (the property always points to classes) and finishing at 0.5 (the property point to classes half of the times), with decrements of 0.01 for each test.

In this chapter, we do not include the results of all these configurations, but just the one that we found optimal, i.e., that aligns better with the reference rankings. The best ratio for Wikidata has been $r = 0.99$. The resulting list of class-pointers with $r = 0.99$ is shown in Table 3.4.

In this experiment, we seek to rank every Wikidata element that fits in Wikidata’s definition of what a class is. Then, since the set of classes is known a priori and provided to the algorithm, the configuration of the security threshold should be $\theta = 0$.

¹⁶http://data.weso.es/classrank/wikidata/wikidata_classpointers_ratio.json Accessed in 2022/05/03.

Pos.	Human Hosts		Every Host	
	Class	Mentions	Class	Mentions
1	Q5- human	312,245	Q5- human	26,902,123
2	Q55983715- organisms [...]	261,714	Q55983715- organisms [...]	3,296,393
3	Q515- city	215,273	Q6256- country	1,651,146
4	Q644371- international airport	209,245	Q3624078- sovereign state	1,553,641
5	Q6256- country	32,584	Q13442814- scholarly article	1,341,577
6	Q3624078- sovereign state	30,257	Q16521- taxon	1,306,359
7	Q28640- profession	21,421	Q11424- film	1,301,887
8	Q7270- republic	13,213	Q3947- house	1,279,807
9	Q12737077- occupation	11,975	Q48264- gender identity	1,231,213
10	Q20181813- colonial power	9,960	Q4369513- sex of humans	1,228,445
11	Q63791824- Baltic Sea countries	9,111	Q427626- taxonomic rank	1,013,172
12	Q43702- federal state	8,813	Q340169- communication medium	955,023
13	Q11173- chemical compound	8,519	Q3100180- rank	908,404
14	Q619610- social state	8,297	Q13578154- rank	907,632
15	Q4209223- Rechtsstaat	8,221	Q3331189- version, ed., or translat.	826,798
16	Q15079663- r. t. railway line	8,072	Q10876391- Wikipedia lang. edition	791,432
17	Q45- Portugal	7,548	Q515- city	773,382
18	Q11900058- explorer	7,493	Q7432- species	741,928
19	Q13442814- scholarly article	6,423	Q16970- church building	702,134
20	Q48264- gender identity	6,214	Q253019- Ortsteil	699,386

TABLE 3.5: Top20 elements for HH and EH entries in Wikidata

Reference rankings - The most important classes in Wikidata according to class usage with EH and HH entries are shown in Table 3.5. When comparing these rankings with the rankings produced by each technique, we have discarded some special nodes from Wikidata, which are important from a structural point of view due to Wikimedia’s organizational model, but rarely used by end-users in SPARQL queries. The discarded nodes include elements such as ‘Q56005592 - Wikimedia help page’, ‘Q35252665 - Wikimedia namespace’ and some other Wikimedia organizational elements.

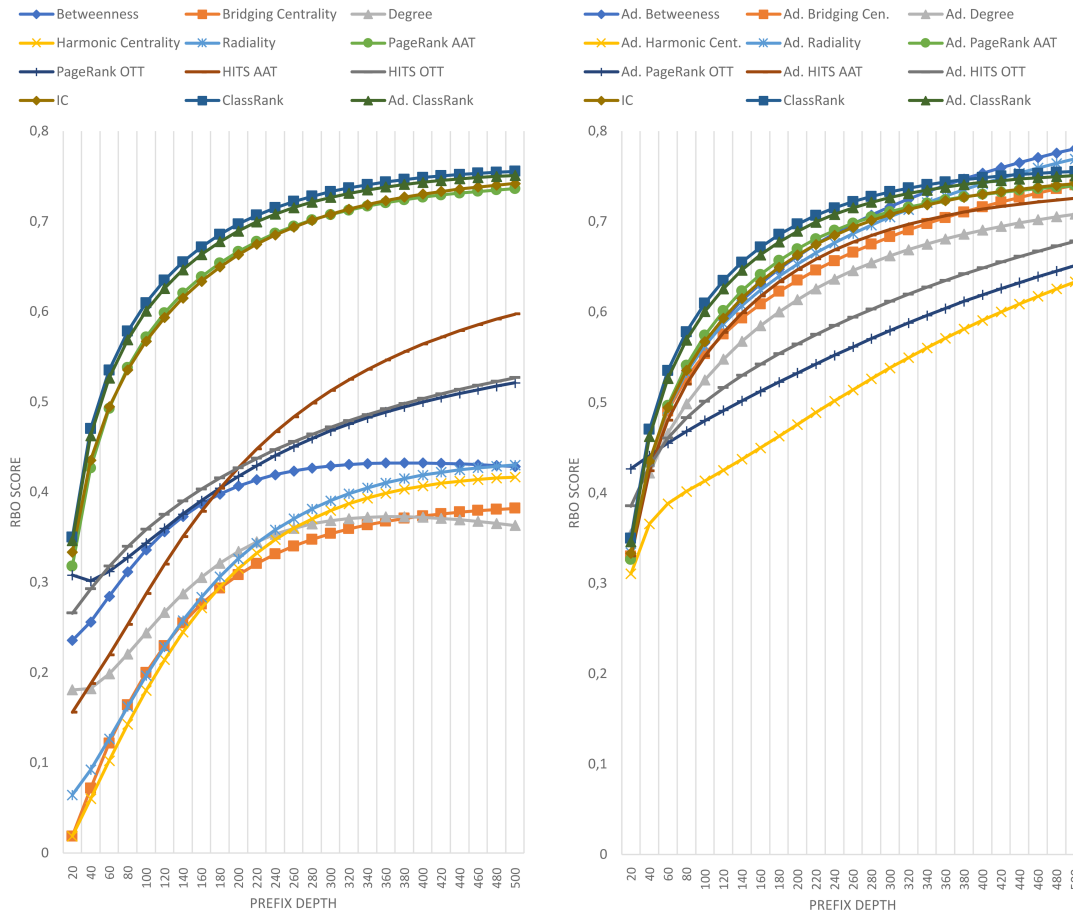
3.3.3 Results

To evaluate the similarity of two rankings R and L using different weights for their top positions, we evaluated several configurations of p in $RBO(R, L, p)$. We used Equation 3.9 to obtain p values for different prefix sizes with a fixed importance of 0.9. In every case, the margin error to calculate p with the script described in section 3.3.1.2 was set to $\alpha = 0.00001$.

Note that each p configuration solves Equation 3.9 for several pairs of $W_{RBO}(1 : d)$ and d . For instance, $p \simeq 0.876343$ is an adequate value for $d = 10$ and $W_{RBO}(1 : d) = 0.9$, but it is also valid for $d = 6$ and $W_{RBO}(1 : d) = 0.8$. We fixed $W_{RBO}(1 : d) = 0.9$ just to provide understandable values of p instead of testing arbitrary increments.

In this chapter, we have run RBO comparing the reference rankings against the rest of the metrics with several d values for $W_{RBO}(1 : d) = 0.9$. Starting at $d = 20$, we have performed comparisons incrementing d in 20 spots each time until the arbitrary depth of 500, whenever this was possible. However, the relatively low number of positions in DBpedia’s HH ranking requires using a smaller maximum depth to explore. When building the reference rankings, some elements receive the same number of mentions. These ties, as explained in section 3.3.1.2, can cause that the number of ranks in a ranking can be lower than the number of total elements.

FIGURE 3.1: Comparison of techniques against EH log in DBpedia.



The total number of classes to rank in the DBpedia ontology is 827. However, with HH entries, there are many ties due to classes with few mentions or no mention at all in the logs (there are unmentioned 583 elements). This situation produces a ranking with just 62 spots.

For this reason, just for the case of DBpedia's HH entries, we start the evaluation at the minimum prefix length of 10 and made increments of 5 spots until the arbitrary depth of 60.

The results of comparing the metrics against the importance ranking of DBpedia's EH logs are shown in Figure 3.1. The results against DBpedia's HH logs are shown in Figure 3.2. The results against Wikidata's EH logs are shown in Figure 3.3. Finally, the results against Wikidata's HH logs are shown in Figure 3.4.

Each figure contains two charts. On the left side, we show the result of the techniques mentioned in section 3.2 in its raw version. On the right side, we show the adaptations of those techniques with IC scores according to the formula explained in Section 3.2.3. The top-performing metrics of each group are included in both sides to show the distance between any technique and the best results obtained. IC is shown on the charts of the right side because the comparison between IC and any adapted technique is quite relevant. As we have already mentioned, the adaptation to AAT of a technique T normalized in $[0, 1]$ consist of a technique T' that assigns to each class c_i a score $T'(c_i) = \frac{T(c_i) + IC(c_i)}{2}$. Then, any T' whose results are consistently worse than IC should be avoided. In such a case, IC scores are performing better by themselves than being used as a factor in T' , even when the computational cost of T' is higher than IC's cost.

FIGURE 3.2: Comparison of techniques against HH log in DBpedia.

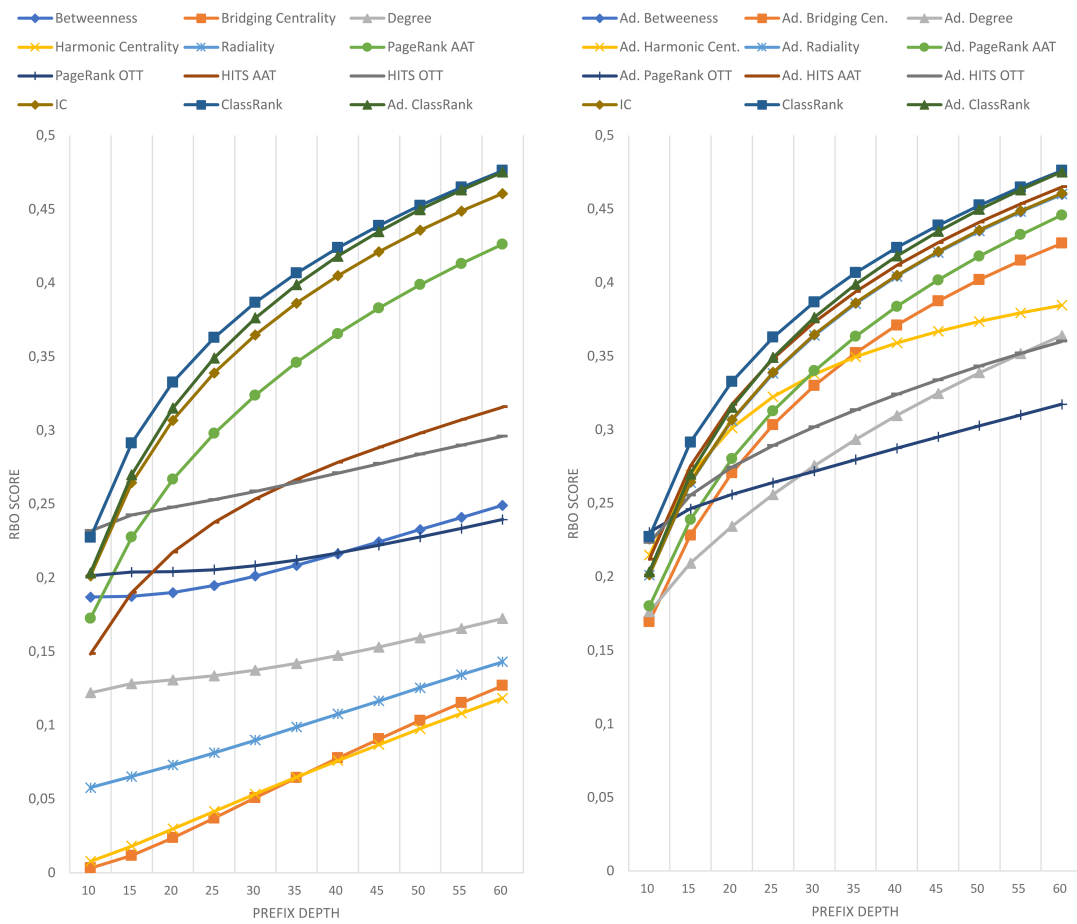


FIGURE 3.3: Comparison of techniques against EH log in Wikidata.

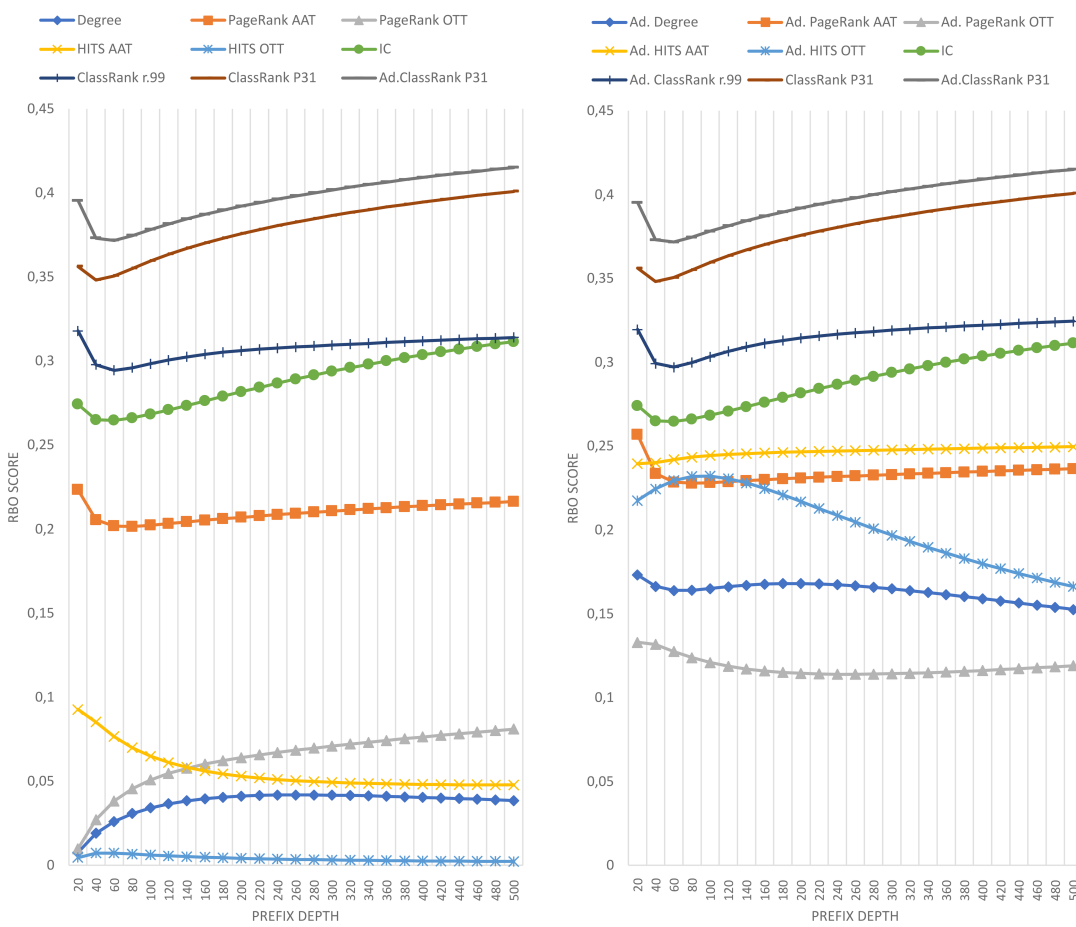


FIGURE 3.4: Comparison of techniques against HH log in Wikidata.

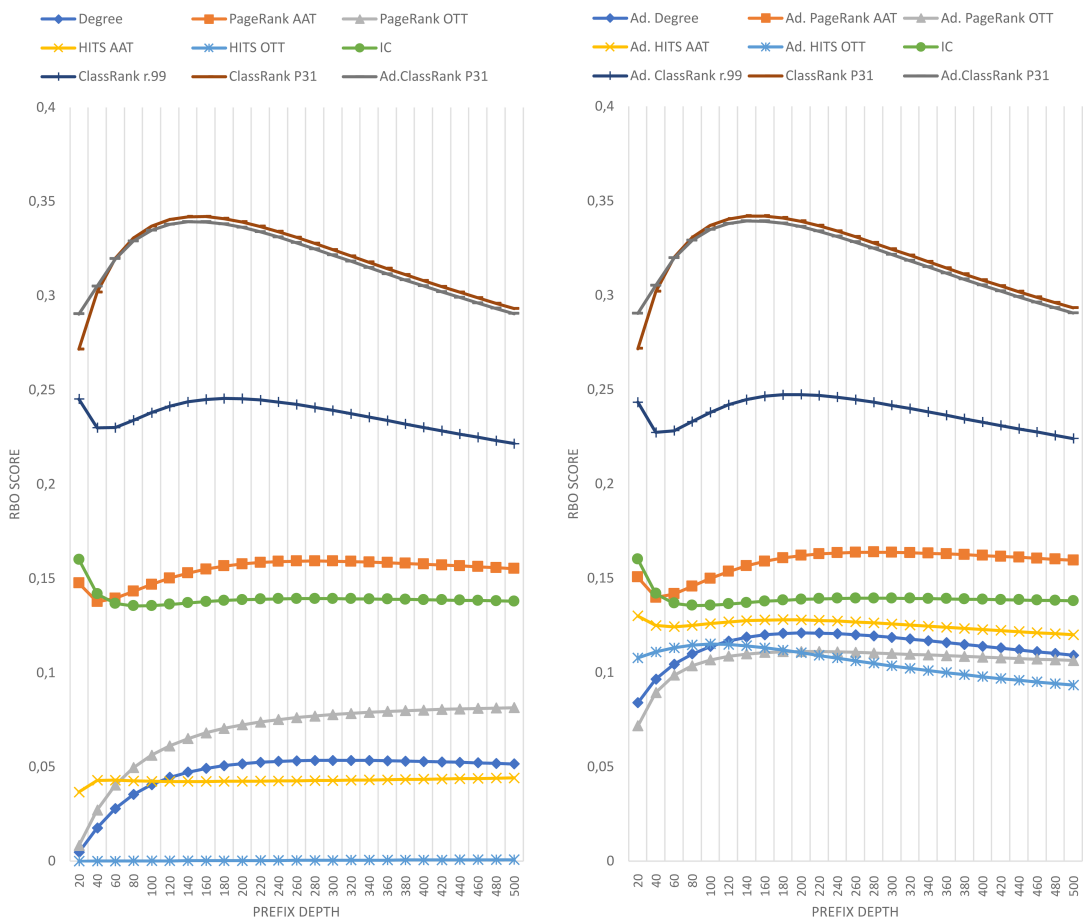
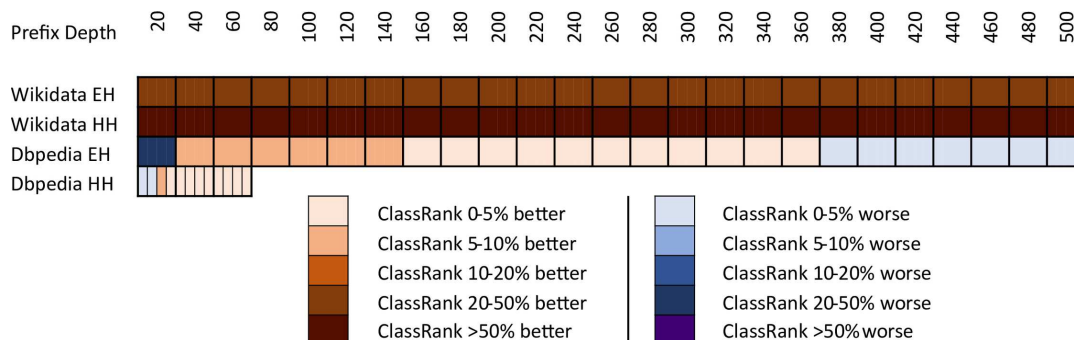


FIGURE 3.5: Performance comparison of ClassRank against any other non-ClassRank metric at every source and prefix depth.



The Wikidata results shown in figures 3.3 and 3.4 include two different executions of ClassRank. We have named *ClassRank P31* the configuration in which $P_C = \{ 'P31 - instance of' \}$, and *ClassRank r.99* the configuration in which P_C includes the properties shown in Table 3.4.

In this section, we have focused on the comparison by means of RBO between the metrics' results and the reference rankings at different prefix depths. To allow further analyses, all the rankings produced are publicly available¹⁷.

3.4 Discussion

3.4.1 Best performing techniques

As can be seen in Figures 3.1, 3.2, 3.3, and 3.4, there is not a metric outperforming all the rest in every case. However, ClassRank and Adapted ClassRank are the approaches performing better at mostly any source and prefix depth configuration.

According to [223], when comparing two techniques, improvements of less than 5% could be discarded and attributed to the nature of the samples chosen in the experiments; improvements between 5% and 10% are noticeable; improvements greater than 10% can be considered material.

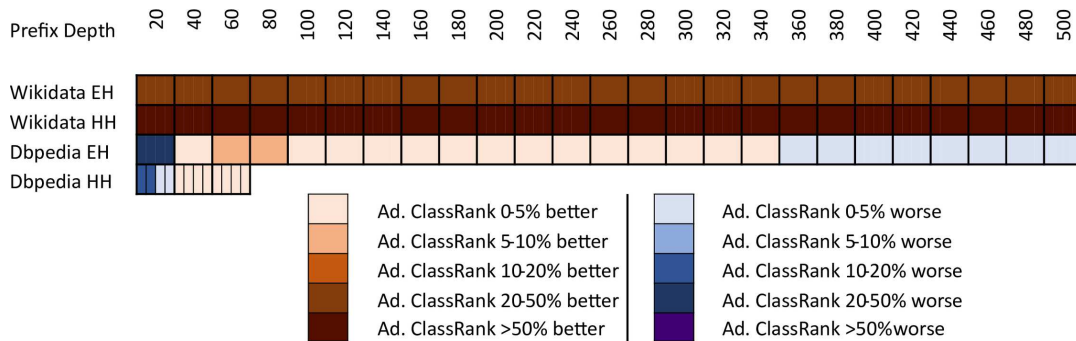
In Figure 3.5, we compare ClassRank and the best non-ClassRank score (i.e., any score which was not produced by ClassRank or Adapted ClassRank) for each source and prefix depth. The ClassRank performance curves used to make comparisons in DBpedia logs are the ones labeled with '*ClassRank*' in Figures 3.1 and 3.2. The curves used for Wikidata are the ones labeled '*ClassRank P31*' in Figures 3.3 and 3.4. In Figure 3.6, Adapted ClassRank scores are compared with the best non-ClassRank score for each source and prefix depth.

With the criteria proposed in [223], one can see that both ClassRank and Adapted ClassRank perform materially better than any other metric for every depth explored in Wikidata logs. This is even more noticeable for HH Wikidata. As shown in Figure 3.4, for depths between 40 and 260, both metrics have an improvement greater than 100% over the rest of the approaches.

With DBpedia logs, even if ClassRank variants are the best-performing ones at most of the depths, this advantage is not that clear. As one can see in Figure 3.1, with EH entries, Adapted PageRank OTT performs materially better than ClassRank until depth 20. Between depth 40 and 140, ClassRank performs noticeably better than the

¹⁷<https://github.com/DaniFdezAlvarez/classrank/tree/develop/experimentation/doc/> Accessed in 2022/05/03.

FIGURE 3.6: Performance comparison of Adapted ClassRank against any other non-ClassRank metric at every source and prefix depth.



rest of the approaches. From this depth, ClassRank performs slightly better than the second best-performing approach, which is Adapted PageRank ATT. However, the distance with this curve is always inferior to 5%. In depths beyond 380, ClassRank is slightly outperformed by Adapted Betweenness, and beyond 420 by Adapted Radiality as well.

We can observe similar circumstances with DBpedia’s HH entries. ClassRank is slightly outperformed by Adapted PageRank OTT at depth 10. Then, ClassRank performs noticeably better than the rest of the approaches at depth 15. From this point, ClassRank keeps being the best-performing approach, but with advantages lower than 5% over the second one at every depth.

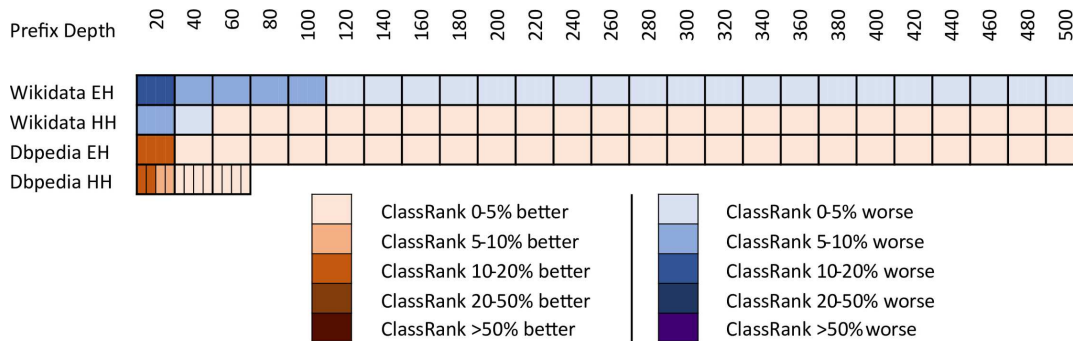
The high top of DBpedia’s rankings are the only regions explored in which any technique noticeably outperforms ClassRank. That situation happens with $d = 10$ for HH entries and with $d = 20$ for EH entries. Several factors related to the topology of DBpedia Ontology cause this.

On the one hand, this ontology is structured as a tree in which *owl:Thing* is the root and has many direct children. Many classes do not have any children at all and, when they do, their subtrees are not deep. This structure causes *owl:Thing* to be ranked in the first position of most OTT metrics, with a great distance to the class ranked 2nd. The scores of the next classified classes are much closer. On the other hand, the IC scores have higher score differences in the top positions of the rank. Then, when the OOT metrics are adapted, except for the rank of the *owl:Thing* class, IC has a more significant influence on the top spots of the adapted metrics. Also, among those classes that do have a populated subtree, we can find elements such as *dbo:Place*, *dbo:Organisation*, or *dbo:Person*. These nodes are general enough to have a subtree of classes, boosting its rank in OTT approaches, such as Degree or PageRank. At a time, these nodes are direct classes of many instances, which increases the chances of being mentioned in SPARQL logs.

These factors, combined with the relatively small number of elements in the DBpedia Ontology, make some adapted approaches to perform even better than ClassRank at the very top of DBpedia’s rankings. However, this works just for few nodes, such as the mentioned ones. The deeper we go into the ranking, the better perform ClassRank and, in general, the metrics which were originally AAT.

It is hard to determine in which conditions Adapted ClassRank performs better than ClassRank and vice-versa. In Figure 3.7, we show the comparison between ClassRank and Adapted ClassRank. As one can see, the general case is that ClassRank outperforms its adapted version. Nevertheless, in most of the cases, the relative difference between these two approaches is lower than 5%. The only exceptions to these situations occur in the top positions of each reference ranking. Adapted

FIGURE 3.7: Performance comparison of Adapted ClassRank against any other non-ClassRank metric at every source and prefix depth.



ClassRank performed materially or noticeably better than ClassRank until the top 100 of Wikidata EH, and until top 20 for Wikidata HH. In opposition, ClassRank performed materially or noticeably better until top 20 of both DBpedia EH and HH. With this data, it cannot be concluded whether ClassRank outperforms Adapted ClassRank in general terms.

3.4.2 General performing of techniques in different sources

Most of the scores of any technique for any depth and source are, in general, far from 1, which would mean a perfect alignment between a reference ranking and the ranking produced by a given approach. The highest similitude between any reference ranking and any tested metric is reached by Adapted Betweenness with DBpedia's EH entries, scoring 0.781 for a prefix depth of 500. However, except for the case of DBpedia EH curves, no other reference ranking produces any measure above 0.5 at any checked prefix depth.

That means that there is not such a technique able to precisely capture the notion of importance observed in log's class usage. The probable cause of this is that there is not either a perfect mathematical relation between a graph's structure and the actual usage of its nodes in a SPARQL endpoint. Nevertheless, log usage information is not always available, so it is worth it to keep working on metrics able to find the best possible alignment between a graph's structure and its actual usage.

With DBpedia, the lowest scores occur at the top positions of the rankings. The higher is the ranking prefix, the better is the final score. All the performance curves in Figures 3.1 and 3.2 have a linear or asymptotic-like shape with a constant improvement. That can be explained by the fact that the number of classes to rank is nearly covered by the max prefix length chosen. Then, when exploring regions of the ranking deep enough, even by pure randomness, it is easier to find classes shared by the reference list and the list under evaluation. That boosts RBO scores when considering wider prefixes.

In opposition, a prefix depth of 500 represents just 0.0002% of the classes detected in Wikidata. Then, the chances of getting shared classes between the list compared due to randomness dramatically decrease. With Wikidata's large number of classes to rank, there is not a general tendency with performance curves. Even within the same chart, we observe linear, asymptotic, and parabolic-like curves. As shown in Figures 3.5 and 3.6, with many classes to rank, ClassRank materially outperforms any other metric at every measured prefix depth.

3.4.3 EH vs HH results

It seems that, in general, the correlation between the graph's structure and the class usage in SPARQL endpoints is higher with robotic agents. For a given metric, prefix depth, and source, the general case is that the HH score is lower than the EH score. Also, the general case is that the difference between these two scores is material, i.e., higher than 10%.

However, as discussed in the previous section, ClassRank and Adapted ClassRank outperform the rest of the techniques both for EH and HH entries. This is, among the evaluated algorithms, ClassRank seems to be the approach that captures better the notion of importance w.r.t. class usage with both types of traffic (organic and robotic).

3.4.4 OTT, AAT and adapted metrics

With the notion of class importance adopted in this paper, the general case is that AAT techniques outperform OTT ones. There are very few exceptions to this observation. The most salient ones happen 1) with DBpedia HH entries, where HITS OTT outperforms HITS AAT until depth 35, and 2) with Wikidata, where HITS AAT scores are worse than some OTT approaches for some prefix values. This last exception occurs from depth 140 with EH entries and from depth 80 with HH entries.

Not computing the A-BOX section of the graph led to a loss of valuable knowledge about the KG's topology. The adaptation of OTT metrics with IC scores is proposed to use, in a computationally cheap way, a salient feature of the graph, which are class-instance relations. It is remarkable that the general case for every raw metric T is that its adapted version T' outperforms T . That is probably because IC scores perform better than almost any other raw metric. The only exception to this is ClassRank, which is also the only one that is not clearly outperformed by its adapted version.

It is worth mentioning that, in every case, ClassRank and Adapted ClassRank tend to describe performance curves of similar shape and, except for the top positions and the Wikidata EH log, they have score points nearly swapped. That means that, in general, the adaptation with IC scores does not have a noticeable impact over ClassRank. The probable reason for this is that several top-ranked classes appear in the tops of both IC and ClassRank rankings. Since the top positions of the rankings have a determinant impact on RBO scores, the inclusion of IC's well top-ranked elements boosts the results of most of the metrics. That improvement does not happen when adapting ClassRank because those key classes are already top-ranked in raw ClassRank.

3.4.5 Configuration of class-pointers

The possibility to configure the set of class-pointers P_C to be used during the execution of ClassRank is more linked to the concept of relevance than importance, as it is a user's choice to let the algorithm focus on some specific aspects of the KG.

We have been able to evaluate different P_C settings with Wikidata. However, due to the high number of potential class-pointers in this source, we have not tested every possible configuration of properties, but just those described in section 3.3.2.2. As can be seen in Figures 3.3 and 3.4, even the best P_C setting performed materially worse than the straight-forward configuration $P_C = \{ 'P31 - instance of' \}$. Custom and more user-centered configurations of P_C could achieve better results.

Technique	Complexity
Degree	$O(n + e)$
Betweenness	$O(n \cdot (n \cdot e))$
Bridging Centrality	$O(n \cdot (n \cdot e))$
Harmonic Centrality	$O(n \cdot (n + e))$
Radiality	$O(n \cdot (n + e))$
Instance Counting	$O(n + e_i)$
PageRank	$O((n + e) \cdot \frac{\log n}{\epsilon})^*$
HITS	$O(e \cdot k)$
ClassRank	$O(n_c + e_i + (n + e) \cdot \frac{\log n}{\epsilon})$

n = number of nodes; e = number of edges; e_i = number of instantiation edges;
 n_c = number of classes; k = maximum number of HITS iterations

* According to [224], PageRank can be computed in ϵ rounds, being $\frac{\log n}{\epsilon}$ the reset probability of PageRank, and having each round a complexity of $O(n + e)$

TABLE 3.6: Computational cost of the techniques evaluated

Also, it is worth mentioning that even if *ClassRank r.99* performed worse than *ClassRank P31*, both *ClassRank r.99* and *Adapted ClassRank r.99* performed better than any other raw or adapted approach (excluding other *ClassRank* configurations).

3.4.6 Computational cost vs performance

In Table 3.6, we included the computational cost of every technique evaluated.

As one can see, Degree and IC are the cheapest techniques to execute, as they perform simple counting actions over the target nodes.

Implementations of PageRank and HITS are usually based on eigenvector computations. Even if the complexity of this is linear w.r.t to the number of nodes and edges, both approaches require several iterations until they converge and reach a result, increasing the complexity compared to Degree and IC. *ClassRank* has a complexity similar to PageRank because it requires PageRank scores. Once those scores are computed, the complexity of the rest of the algorithm is linear w.r.t. the number of nodes and target edges.

Betweenness, Bridging Centrality, Harmonic Centrality, and Radiality have complexities higher than quadratic w.r.t. numbers of nodes, as they all need to compute the shortest path between any pair of nodes in the target graph. This complexity makes them suitable for small or moderate-sized structures, but they can have a prohibitive performance cost for big networks.

The adapted techniques have not been included in Table 3.6 because they all have the same computational complexity as their raw version. That is because the adaptations purely consist of normalizing and averaging the original scores with IC scores, and IC has the lowest computational cost of all the analyzed techniques.

When choosing a metric to measure class importance, both the performance and the computational cost should be considered. Different contexts may lead to different decisions regarding whether it is preferable to prioritize performance or cost. However, some techniques could be discarded when they perform worse than their competitors w.r.t. to both decision parameters.

Even though IC is the simplest approach, it outperforms any other raw technique except *ClassRank* and, as one can see in Figure 3.1, PageRank AAT in a section of

Rank	ClassRank	Instance Counting
1	dbo:Person	dbo:Person
2	skos:Concept	dbo:CareerStation
3	dbo:CareerStation	dbo:SportsTeamMember
4	dbo:Settlement	dbo:Settlement
5	dbo:SoccerClub	dbo:PersonFunction
6	dbo:SoccerPlayer	dbo:Village
7	dbo:SportsTeamMember	dbo:TimePeriod
8	dbo:Country	dbo:Album
9	dbo:City	dbo:Insect
10	dbo:PersonFunction	dbo:SoccerPlayer
11	dbo:Village	dbo:Film
12	dbo:AdministrativeRegion	skos:Concept
13	dbo:TimePeriod	dbo:OfficeHolder
14	dbo:Insect	dbo:Company
15	dbo:Album	dbo:Plant
16	dbo:OfficeHolder	dbo:MusicalArtist
17	dbo:Film	dbo:Single
18	dbo:Company	dbo:Building
19	dbo:MusicGenre	dbo:Town
20	dbo:MusicalArtist	dbo:Athlete

TABLE 3.7: Top 20 of ClassRank and Instance Counting

DBpedia EH ranking. Also, it is worth mentioning that, in most of the cases, the adapted version of those algorithms does not outperform IC scores. Again, the only exception to this is Adapted ClassRank and, as it is shown in Figures 3.1 and 3.2, PageRank AAT and HITS AAT for some sections of the DBpedia logs.

Since the complexity of HITS, PageRank, and ClassRank is similar, but ClassRank seems to outperform HITS and PageRank, we will discuss the differences between ClassRank and IC. In table 3.7, we show the top 20 elements in DBpedia according to ClassRank and IC.

As one can see, the rankings contain elements related to very similar domains (mainly arts, sports, geopolitical divisions, and people). Fourteen classes appear at the top-20 of both lists, and some of them even have an identical rank. That makes sense, as many incoming links coming from many instances ensure high importance with IC, but, frequently, they also imply high importance with ClassRank due to accumulated PageRank scores of that many instances.

A couple of missing elements in the top-20 of IC raises a clue about the kind of classes in which these two techniques heavily disagree: *dbo:SoccerClub* and *dbo:Country*. With ClassRank, *dbo:SoccerClub* ranks 5th and *dbo:SoccerPlayer* ranks 6th. The different soccer players are connected to their club, so some of the importance accumulated by all soccer players goes to their respective teams. With this, even if there are much fewer instances of clubs (21,955) than players (117,619), these two classes can achieve a similar score of importance. In contrast, *dbo:SoccerClub* descends to position 34th with IC.

The case of *dbo:Country* is more revealing. Instances of countries are frequently key elements to link different topics in cross-domain KGs such as DBpedia. Many kinds of individuals can be linked to their country, such as smaller administrative divisions, people, geographical entities, or events. With ClassRank, that accumulated

importance is good enough to rank 8th. *dbo:Country* is one of the top seeds according to the reference rankings as well (9th with HH and 21st with EH). In contrast, *dbo:Country* descends to 145th with IC.

Similar examples are *dbo:MusicalGenre* (19th with ClassRank vs 223rd with IC), *dbo:Legislature* (42nd vs 192nd), and *dbo:Language* (24th vs 85th). In general, we can say that IC and ClassRank produce rankings that tend to be quite similar. However, ClassRank can capture the importance of classes that do not have too many instances when those instances are important elements of the KG. In contrast, IC cannot capture the importance of such classes.

3.5 Related work

We divide the related work section of this chapter in two subsections.

In the first one, we will focus on those proposals that explore node importance as a main goal or as a previous step to achieve another goal, commonly graph summarization. It is important to mention that node importance is not essential for every graph summarization technique though. For instance, some alternative approaches are based on pattern-mining methods [225] or quotient summaries [226, 227]. However, we will not analyze such proposals in detail as they fall out of the scope of this thesis.

In the second subsection, we will review algorithm proposals that are adaptations of PageRank. None of these proposals have been evaluated in the experiments of this chapter due to several reasons. Most of them are designed to work as a relevance metrics instead of an importance one. Some others need an input or raise an output which is not compatible with our domain of application. However, they are relevant to our work as they inspire some of the ClassRank's basic ideas.

3.5.1 Scoring entities or classes in graphs

Several authors have already used centrality metrics to determine entity importance or relevance in KGs.

In [28], a study of different techniques to detect class importance is performed. They check the performance of Degree, Betweenness, Bridging Centrality, Harmonic Centrality, Radiality and Ego Centrality against a gold standard built using DBpedia logs, in the same manner that we do in this chapter. Also, they propose the adaptation of the aforementioned metrics to make use of instance information which has been evaluated in our document. The study is a preliminary stage in order to support graph summarization processes. There are two main differences between this study and ours. First, the authors do not experiment with spectral measures such as PageRank. Second, they use Spearman correlation coefficient to determine the similarity between the rankings, so the top and the tail of each ranking have the same weight on the results.

Authors in [211] perform another general study of class importance as a previous step for ontology summarization. All the techniques used are applied over the ontologies, i.e., they do not use A-BOX knowledge at any point.

In [212], another case of measuring class importance for ontology summarization is presented. They propose two methods, one inspired in Degree, which is defined in [228], and another one inspired in Closeness. However, those methods require some user input, such as relevant domain-specific relations in ontologies or weights for certain elements.

RDFDigest+ is a tool to perform RDF/S Knowledge Base exploration using summaries [213]. It allows the user to choose and combine many different centrality algorithms to identify the most important nodes. It also uses information related to the instances of each schema element. In [229], the same authors expose *zoom* and *extend* operations for RDFDigest+, which enable the user to get ontology summaries with different detail level in an efficient way. They perform a stage of class importance detection in which HITS, PageRank, and Betweenness are used. They also apply the adaptation of these approaches with IC scores proposed in [28].

In [215], an approach to rank relevant elements in RDF graphs is presented. The authors define an approach that produces node scores with a hybrid strategy. On the one hand, they compute structural relevance by using an adaptation of Degree where the relations can be weighted according to user preferences. On the other hand, they compute frequency of usage nodes in SPARQL query logs. They propose two graph summarization techniques built on top of these notions of node relevance, named *SummaryKG*, which only uses structural information, and *Query-sumKB*, which uses log frequencies too.

Another approach for graph summarization called *WBSum* is presented in [214]. In order to locate important nodes for a certain summary, the authors also use a hybrid approach in which they locate important nodes w.r.t. usage in SPARQL queries combined with a notion of structural importance based on Personalized PageRank [2].

In [230], the authors perform a study of several cross-domain KGs quality, including DBpedia and four more sources. One of the features studied is class coverage for different knowledge domains. They manually classify each class to belong to the different domains. Then, they measure the importance that each class provides to each domain by counting their instances.

In [231], PageRank is applied over a graph of Wikipedia linked entries. Each entry is represented by its DBpedia URI, so they produce a ranking of DBpedia entities based on the Wikipedia link graph. In combination with other methods, the results are used for entity summarization [232, 233]. In order to merge the information of different Wikipedia chapters, the authors compute a ranking of Wikipedia entities using their Wikidata URI, which is unique for all the languages. In these works, PageRank is used as a base metric to rank entities in a KG. However, they use a voting method w.r.t. different Wikipedia chapters. Thus, this technique cannot be applied over KGs whose elements are not linked (directly or indirectly) to Wikipedia pages.

In [234], an approach to rank classes in DBpedia is presented. As well as ClassRank, this work is also based on aggregation of PageRank scores. Nevertheless, these scores are not obtained from DBpedia's structure, but each entity receives the PageRank score of its associated page in Wikipedia. Then, this technique cannot be applied over KGs whose entities are not linked to Wikipedia. Also, the authors combine the aggregation of PageRank scores with some other parameters such as Instance Counting.

In [235], PageRank is applied over the DBpedia link structure to mine significant concepts. Given an element in DBpedia, the authors track its most related concepts by exploring its neighborhood in the graph, and they rank those results according to inverse PageRank. They consider that the most related elements are the ones with a lower PageRank score. The authors argue that elements with low PageRank are not so well connected because they are too specific of a given topic. Hence, those URIs in the neighborhood of a concept c with low PageRank may have higher chances

of being semantically closer to c than those with high PageRank, which may be too transversal.

An approach to explain and use entity relatedness in KGs is presented in [236]. The authors formalize the concept *relatedness explanation* between two entities as a subgraph containing paths that link those two entities. Once they obtain an explanation, they can detect pairs of entities in the KG sharing a similar notion of relatedness. This proposal focuses on the detection and ranking of important paths between nodes. The authors compute the importance of each path using mainly properties (predicates) instead of entities. However, there is also a stage of ranking entity importance which is PageRank-based.

In [237], an approach to identify key concepts in ontologies is presented. This work uses a notion of importance based on experts' agreement. The algorithm presented combines purely topological features of the ontology with cognitive concepts such as natural categories [238] or popularity according to results in web search systems. The approach tries to detect the elements that best summarize the semantics of the target ontology. The experiments show a high correlation between the approach's results and the experts' choices. This work differs from ours in their human-based notion of importance and the type of ontologies evaluated, which are small to moderate-sized (the biggest ontology used in the experiments contains 247 elements), as well as domain-specific.

Freebase associates a score with ranking purposes to each one of its stored entities. However, this score is not computed with PageRank, but using a simpler formula based on link counts of an entity in Freebase KG and its associated page in Wikipedia [239].

Wikidata Project maintains some special pages offering some metrics of the graph that are frequently updated¹⁸. Among these results, link counts of the most used elements can be found, but there are no reports about class importance or PageRank-like scores of any element.

3.5.2 Alternative centrality measures based in PageRank

The idea of using personalized versions of PageRank to adapt the algorithm to different contexts was early suggested in [2]. The original authors of PageRank propose an adaptation called Personalized PageRank (PPR). In PPR, there is a set of restarting nodes which are the only ones that the random walker can jump to when it gets bored of following links. PPR is still widely used, and several works propose ways to optimize its execution [240].

Since that proposal, many PageRank adaptations have been published. Probably, the closest adaptations to our domain are those that compute aggregations of PageRank scores. A representative example of this strategy is BlockRank [241], which divides the target graph into several disjoint blocks of smaller units. An illustrative use of BlockRank-like strategies is HostRank [242], which was thought to be applied over web structures. Nevertheless, most BlockRank-like approaches are not compatible with our domain. In RDF graphs, an entity can be an instance of several different classes. Hence, hypothetical blocks formed by instances of the same class would not be disjoint.

Most of the PageRank adaptations have been thought to measure the relevance of an element in a KG w.r.t. a query [243]. Those approaches are focused on IR tasks and tend to rank entities using notions of semantic relatedness between query and

¹⁸https://www.wikidata.org/wiki/Wikidata:Database_reports Accessed in 2022/05/03.

resource. Some of them measure importance and are used in combination with other notions to produce some result [244, 245]. Some others measure relevance, including strategies such as text similarity or exploration of topic sub-graphs in the algorithm itself [246–250].

OntologyRank [244] is designed to rank Semantic Web Documents (SWD), such as ontologies or RDF files, which are linked to each other through their internal elements. The algorithm uses the semantics of the properties to divide them into four different categories. Then, it computes a version of PageRank where each link can be weighted w.r.t. each category. Although it follows a strategy of aggregation of PageRank-like scores, OntologyRank is designed to rank different SWDs instead of elements within a single SWD.

PopRank [245] is an adaptation of PageRank designed to be applied over a network of objects. It combines two factors to obtain the popularity of an object: a weighted PageRank in which every property has its own weight, and the PageRank of the database/web page which contains the object (Web Popularity). PopRank is thought to assign a score to every entity in the graph, i.e., there is no aggregation or grouping of individuals in some class or cluster, so PopRank and ClassRank have different domains of application. Also, PopRank has a stage in which some training data should be provided by experts, which may be too costly in graphs with many properties such as DBpedia or Wikidata.

ReConRank [246] is a PageRank adaptation designed to be applied over RDF domains that combines the approaches of ResourceRank and ContextRank. ReConRank is closely related to search and retrieval domains. The ranking of entities is not applied over the whole target graph but over a sub-graph composed of certain elements that are related enough to some keywords. The scores produced are a measure of relevance w.r.t. a query.

RareRank [247] makes use of transition scores between entities, as well as PageRank does. However, it proposes a Rational Research model to define transitions between elements, aiming to simulate a human strategy of jumping from one document to another. RareRank is thought to be applied in semantic search of research documents. It relies on meta-data associated with scientific papers modeled in an ontological way, as well as topic relatedness computed with Latent Dirichlet Allocation [251]. As with ReConRank, RareRank produces scores of relevance instead of importance. Also, the model of Rational Research should be adapted to apply it in domains different from scientific documents.

DBpediaRanker [248] describes an algorithm to rank DBpedia entities w.r.t. a query. In this case, the authors do not follow a PageRank-like approach, but they consider several different notions of similarity. This includes textual similarity, proximity to a certain set of seed nodes, or results supported by external resources, such as search engines or tagging systems. Thus, although the main goal is also the ranking of RDF resources, this approach has a specific domain of application and cannot be used to measure class importance.

TripleRank [249] is a HITS-based algorithm to rank entities w.r.t. a subject and a facet (predicate) in RDF environments. TripleRank gives a notion of relevance w.r.t. some other graph elements instead of importance per se.

DWRank [250] ranks ontology concepts in search and retrieval environments. It combines three types of notions to rank a given element: text similarity with a query, hub score within its own ontology using a reversed PageRank function, and authority of its ontology w.r.t. the rest of ontologies. The goal of the algorithm is to rank ontology members, and it works purely with T-BOX elements, i.e., it does not use any instance information to produce its results.

3.6 Conclusions

At the beginning of this chapter, the following Research Question was stated:

- **RQ3:** How can we identify the most important classes of an RDF graph?

To answer it, we have proposed a new algorithm to measure class importance in RDF graphs and compared it with state-of-the-art techniques. Our approach, called ClassRank, assigns each class an importance score which is a number in $[0, 1]$. The ClassRank score of a class c is the probability that a person that navigates a graph starting in a random node and then jumping randomly from node to node following links has to stop at an instance of c . ClassRank is an aggregation of PageRank scores. The score of a class is obtained by adding the PageRank scores of all its instances. The algorithm allow to use a lasso notion of the class-instance relation. To determine which entities are instances of a given class, it allows the user to specify a custom set of properties CP . Every instance e that points to a class c using a property $p \in CP$ is considered an instance of c . This configuration parameter allows for adapting ClassRank to specific user needs in different scenarios.

We evaluated different techniques to measure class importance in RDF graphs based on the graph's topology. To compare the approaches, we have performed experiments using a notion of importance based on class usage in SPARQL logs. We elaborated class rankings sorting classes w.r.t. their number of mentions in the logs and measured the similarity of those rankings with the lists produced by the evaluated techniques. This similarity was measured using Rank-Biased Overlap.

The experiments raise several conclusions. We observed that, in general, ClassRank outperforms the rest of the evaluated approaches in terms of similarity with the reference rankings. Instance Counting, which is the technique requiring less computational time among all the studies approaches, outperforms every other studied approach except for ClassRank and, in some cases, PageRank. We also observed that those approaches considering just T-BOX statements are, in general, outperformed by techniques that compute A-BOX knowledge as well.

It has been proved that a simple adaptation of most of the techniques by averaging its results with IC scores improves the performance of the original technique. The only exception to this is the adaptation of ClassRank scores, for which it is unclear whether the adaptation with IC improves the performance of the base ClassRank algorithm.

A qualitative comparison of ClassRank and Instance Counting shows that they produce rankings with a high proportion of elements similarly ranked. However, Instance Counting is not able to catch the importance of really well-connected elements that do not have too many instances, such as the class *dbo:Country* in the DBpedia ontology.

The evaluated approaches have been compared against rankings of class usage with log entries generated by organic agents and robotic agents. It has been shown that, in general, all the evaluated techniques align better with the machine-related log entries. Nevertheless, in our experiments, ClassRank outperforms the rest of the proposals for both organic and robotic entries.

In the context of our thesis, we determined with this experiments that, among the evaluated techniques, ClassRank is the most suitable approach to detect important topics (classes) in an RDF graphs. However, we have proved that ClassRank is a competitive approach to measure class importance in contexts unrelated to this thesis. Thus, we consider that ClassRank can be useful not just as a software piece

within our thesis, but as a standalone system. To let the scientific community use our algorithm, our Python implementation of ClassRank is publicly available in a GitHub repository which includes some documentation and example code to run it.

3.6.1 Future Work

We consider that ClassRank's maturity and performance is adequate to fulfill the goals of our thesis. However, we have detected several ways to 1) perform further evaluations on ClassRank performance, and 2) improve the algorithm from various points of view.

We consider several lines of future work:

- **Extended evaluations.** We have performed experiments against DBpedia and Wikidata, both cross-domain projects well known for the LOD community. It can be interesting to extend this evaluation to some other sources, both cross-domain or domain-specific. Indeed, we think that domain-specific graphs could be a better environment to evaluate the potential impact of custom sets of class-pointers, as the number of properties to discuss could be lower and related to the source's main use cases.
- **Aggregation of scores different to PageRank's.** ClassRank determines class importance by aggregating the importance of a class's instances. In its current definition, the instance importance is determined by means of PageRank. But it is feasible to think in versions of ClassRank built on top of different entity level importance scores. PageRank for non-directed graphs, HITS, or Degree are some of the options to consider.
- **ClassRank as a relevance metric.** ClassRank could be used as a relevance metric instead of an importance metric in various manners:
 - It could be combined with a blocking technique for limiting the target graph to compute with ClassRank to a subgraph of the original source. This subgraph would just contain elements relevant for a certain query or topic.
 - The set of class-pointers could be customized so it just contains properties which are relevant for a certain context or purpose.
 - The algorithm could incorporate custom weights at several stages: for properties in a set of class-pointers, for entities w.r.t. to some user criteria, etc.

Chapter 4

Mining triples to extract shapes

Note: Most of the content of this chapter has been published in a paper entitled *Automatic extraction of shapes using sheXer* [252].

4.1 Introduction

The main goal of our thesis is being able to propose a system to extract RDF shapes from natural language found in social media. Shapes can be used as machine-readable representations of conceptual knowledge expressed in a text. Indeed, RDF shapes can be informally described as concept descriptions. For example, let us consider the following simplistic definition of the concept *country*: “A country consists of a land area that can limit with some other countries, and it always has a capital city”.

The Shape Expression shown in Figure 4.1 captures most of the information expressed in this quoted definition. It can be feasible to think about a system able to extract the shape in 4.1 from that textual description. An alternative way to extract shapes from text content is the generalization of examples. Let us consider the following statements about different countries:

1. “Spain has 550.990 km². Its capital is Madrid, and it limits with Portugal to the west, Morocco to the South, and France and Andorra to the North”.
2. “The capital of Andorra is Andorra la Vieja. The country has a surface of 468 km² and it limits with France and Spain”.
3. “Iceland is a state island whose capital is Reikiavik. The island has a total surface of 102,775 km²”.

A system that it is told that Spain, France, and Iceland are countries could be able to produce the RDF content shown in Figure 4.2. Then, another system could generalize the relations observed for different *ex:Country* instances in 4.2 so it can produce the shape `<CountryShape>` shown in Figure 4.1.

FIGURE 4.1: Example shape of Country in ShExC

```

1  <CountryShape >
2  {
3  {
4    rdf:type [ex:Country] ;
5    ex:landAreaKm2 xsd:int ;
6    ex:capital <CityShape> ;
7    ex:borderWith <CountryShape> *
8  }
9  }
```

FIGURE 4.2: Content about some countries in turtle.

```

1
2 # Info related to Spain
3 ex:Spain a ex:Country ;
4     ex:landAreaKm2 550990 ;
5     ex:capital ex:Madrid ;
6     ex:borderWith ex:Portugal ,
7                   ex:Morocco ,
8                   ex:Andorra ,
9                   ex:France .
10 # Info related to Andorra
11 ex:Andorra a ex:Country ;
12     ex:landAreaKm2 468 ;
13     ex:capital ex:Andorra_La_Vieja ;
14     ex:borderWith ex:Spain ,
15                   ex:France .
16 # Info related to Iceland
17 ex:Iceland a ex:Country ;
18     ex:landAreaKm2 102775 ;
19     ex:capital ex:Reikiavik .
20

```

We consider that extracting shapes from RDF example generalizations instead of literal text descriptions is a more promising approach due to several reasons:

- We think that content describing facts and relations about specific entities is more likely to be found in social media. Literal concept descriptions such as the one used in the first paragraph of this section are more likely to be found in more formal documents.
- We think that generalizing examples rather than processing descriptions may bring opportunities to capture knowledge that could be hardly found in definitions. Let us consider the notion of *country* in a cross-domain source such as Wikidata. It is quite probable that one may find that important artists are related to the place in which they were born, or important events are associated to the country in which they occur. A shape obtained by example generalization could express expected relations between countries, artists, and events in case this information is found in the example entities. Of course, a shape extractor based on definitions could produce similar constraints as well. However, with this example, it seems odd to produce a canonical country definition which include statements such as “*A country is a place where some artists can be born, and some events may occur*”. A shape obtained by example generalization brings a notion which aims to contain a common ground of the actual usage of the individuals that are supposed to conform with that shape. In contrast, a shape build from concept definitions provides a notion based on a single source of information, which may be more reliable regarding to the essential features of the modeled concept, but less informative w.r.t. the possible relations of that concept with some other types of entities.
- A strategy which aims to obtain a shape from example generalization of RFD triples rather than parsing actual text let us divide the main task into two independent subtasks: 1) producing RDF from text, and 2) producing shapes from RDF triples. This division allows us to propose a final system built on top of existing approaches that partially/completely solves each one of those tasks.

The complete process to convert text into RDF is deeply studied in Chapter 5. In this chapter, we focus in one of the subtasks previously mentioned to obtain such a final system. Specifically, we want to provide an adequate answer for the following research question:

- **RQ2:** How can we produce shapes by mining RDF triples?

As already explained in section 2.6, shape languages are relatively new. When we started to work on this thesis, both ShEx and SHACL were still immature proposals and there was no software nor published scientific work describing how to perform automatic extraction of shapes from RDF content. Indeed, to the best of our knowledge, we produced one of the firsts scientific publications suggesting to perform automatic extraction of Shape Expressions from an RDF source [253]. After such proposal, we developed a library for the task of extracting ShEx content. We published a first prototype evaluated on DBpedia [254] and kept evolving our approach. Nowadays, our prototype has become a mature and highly parameterizable Python library called sheXer, which is able to produce both ShEx and SHACL content [252].

However, the global scenario has changed since that first proposal. Within the context of our thesis, the automatic extraction of shapes from existing RDF data is a subtask to achieve a more general goal. But this problem has become a general challenge for the RFD community too [255], and many other approaches to perform automatic shape extraction has been proposed [112, 256–260].

The usual procedure to create RDF shapes is to have domain experts writing them from scratch. However, this is a costly process which becomes harder to perform in complex scenarios. Let us suppose, for example, that we want to produce a shape associated to each class in Wikidata. As explained in section 3.3.2.2, Wikidata contains millions of classes. Many RDF-skilled domain experts would be needed to write an adequate shape for each class in such a case. Much manual work would be required, as well as many discussions about features of specific shapes due to divergences among the domain experts. The more data, more different agents maintaining the data, and more topic variety has the target source, the more difficult becomes the task of writing shapes.

However, automatic extractors allow for reducing or even removing the cost of producing those shapes. Instead of having humans writing down their mental models to ShEx/SHACL, automatics extractors can produce shapes that fit existing RDF content. That is, while humans write shapes describing how the graph is supposed to be, extractors obtain shapes that describe how the graph actually is. The distance between these two notions may depend on the quality of the data. Some other relevant factors in this process are the existence of hidden features hardly observable in the graph (such as complex disjunction constraints), or the actual capabilities and performance of the extractor used.

Depending on the target scenario and user intentions, the shapes obtained could be used as final products or as drafts. sheXer has already been used in both scenarios in different scientific proposals:

- In [261], a Linked Data portal to describe the National Budget of Chile is presented. This portal is based on checked, high-quality and homogeneous data. To describe the classes found in the data, sheXer is used to automatically produce a shape associated to each one of them. The quality of the data makes that the shapes obtained can be used as final products with no further changes from domain experts.

- In [262], a protocol to add knowledge to Wikidata using human coronaviruses as a running example is described. Such protocol suggests using ShEx during the process, both to guide actions and validate content. The proposal starts by locating some key entities of a target class. Then, sheXer is used to extract a shape that conforms with them all. Finally, this initial shape is customized by domain experts to produce a final schema that will be used to guide the rest of the process.

Also, sheXer has been compared with other tools for automatic shape extraction that are being used by the Semantic Web community [25]. The authors of this study made the following statements about sheXer:

- Among the evaluated tools, it showed the best performance when working with large RDF files.
- It is the only one that support both triplestores and raw RDF graphs as input.
- It is the only one that provides a mechanism to filter content in the resulting shapes.

Automatic shape extractors can be roughly classified in two groups. On the one hand, ontology-based approaches produce shapes based on T-BOX knowledge. On the other hand, instance-based techniques, such a sheXer, explore A-BOX relations as well, aiming to describe classes w.r.t. to the neighborhood observed among its instances.

In this chapter, we describe sheXer, an instance-based approach designed to provide an adequate answer for **RQ2**. We describe and evaluate sheXer not just as a piece of our thesis, but as standalone system that can be used in many other contexts in which automatic extraction of RDF shapes is required. Even if nowadays there are several other approaches apart from sheXer that can perform such a task, our approach has a unique combination of features unseen among the rest of alternatives:

- It performs shape inter-linkage, i.e., it can produce constraints that refer to other shape labels. Most of the alternatives use less specific macros instead, such as `IRI`¹.
- It uses an iterative approach that allows for computing big datasets. There is not a strict relation between the size of the computed KG and the memory consumption, as there is no need to keep in memory the whole graph at any point in the process. Most of the alternatives produce errors when handling datasets whose content cannot be completely allocated in the main memory of the host.
- It assigns a trustworthiness score to each one of the inferred constraints. This score is used to sort, filter, and merge constraints while performing the shape extraction.
- It can natively produce both ShEx and SHACL content.

¹In ShEx, the macro `IRI` stands for any node which is an IRI. In SHACL, `sh:IRI` has an equivalent meaning.

- It can produce direct and inverse constraints, i.e., constraints describing triples in which the target node is used as object instead of subject. This feature is shared with the system presented in [112], which is based on ABSTAT [225].

The contents of this chapter are organized as follows:

- In section 4.2, we provide a general description of sheXer. This includes its architecture and main algorithms.
- In section 4.3, we describe and discuss several experiments to evaluate sheXer's performance.
- In section 4.4, we provide an overview of the multiple configurations of our library, and discuss them to explain how our system can be adapted to different use cases.
- In section 4.5, we describe some other works able to perform automatic shape extraction. We place this content after the system description and experimentation to let the reader fairly compare our approach with the alternative ones.
- Finally, in section 4.6, we describe the conclusions of our work related to sheXer and outline future work lines.

4.2 System description

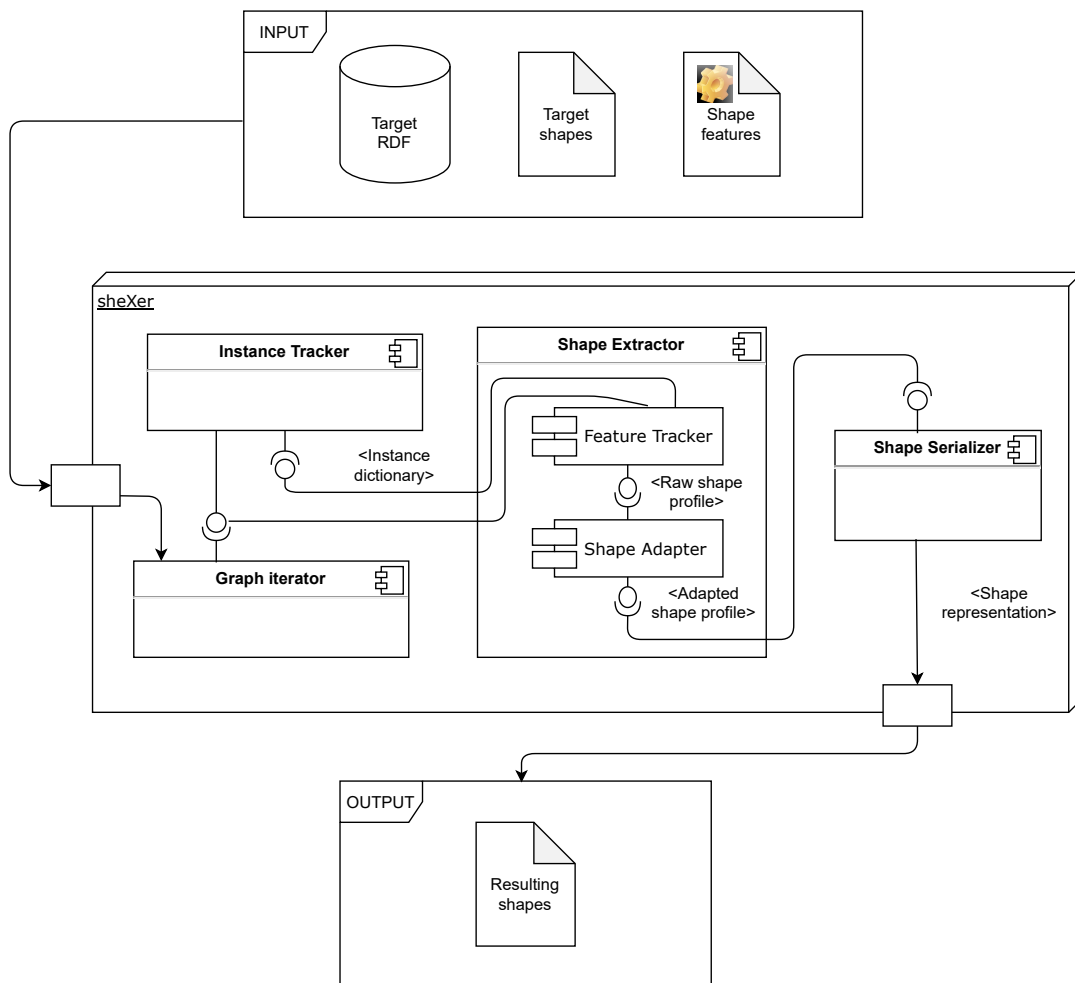
sheXer has been designed with a modularized architecture that allows for adapting the system to many different scenarios. sheXer integrates several modules in a pipeline to produce shapes from input RDF content. Each module expects an input and produces an output. The outputs of each module can be presented to the final user as intermediate results or be consumed by some other module.

sheXer's architecture is shown in Figure 4.3. Its essential workflow consists of the following steps:

- The user chooses some target RDF, target shapes to extract, and (possibly) configures some parameters. The input can be provided as local or remote files, content allocated in main memory, or content exposed in a SPARQL endpoint
- The *Instance Tracker* determines which nodes of the target source will be used to extract which shapes. It consumes relevant triples from the *Graph Iterator*.
- The *Feature Tracker* within the *Shape extractor* generates a set of candidate constraints associated with each shape. It uses the information produced by the *Instance Tracker* and consumes the graph's content using the *Graph Iterator*.
- The *Shape Adapter* filters, adapts, merges, and sorts candidate constraints according to the configuration settings, so a final set of constraints is produced.
- The *Shape Serializer* turns the in-memory shapes produced by the *Shape Adapter* into the content chosen by the user.

Every module can be implemented in several ways, as long as the implementation complies with the specifications of a certain interface. Our current sheXer Python library includes several versions of most of the modules. In the following sections, we detail the structure and mission of each module. Occasionally, we describe some implementation details of our library, as they can be helpful to understand the architecture and the internal details of each module.

FIGURE 4.3: sheXer base architecture.



4.2.1 Graph Iterator

There are two stages of the workflow in which the target RDF source needs to be parsed: 1) determining which nodes will be used to build which shapes, and 2) building the abstract profile for each shape. The target RDF content is served by the Graph Iterator (GI) to perform those actions.

Regardless of the type of input, the mission of the GI is to retrieve relevant triples for those processes. Both stages are performed in an iterative way, i.e., avoiding to host unnecessary pieces of information in main memory whenever it is possible. This allows sheXer to deal with large datasets using inexpensive hardware.

The internal details of this software piece may change according to the kind of input. For example, an RDF input based on local text files can be trivially served by reading small chunks and processing triple by triple. The input could be instead part of the content exposed in some remote SPARQL endpoint, which forces the GI to deal with extra issues. For instance, a different strategy to handle memory could be necessary, as the purely iterative approach could cause too many SPARQL queries being executed against the endpoint. Also, the GI needs to handle errors and timeouts from the SPARQL endpoint.

The current approach of the sheXer library to consume SPARQL endpoints starts by finding which are the seed nodes to build shapes. Once the seed nodes are found, sheXer retrieves all the content in their neighborhood until a certain depth is reached, aiming to perform as less SPARQL queries as possible during this process. After that, sheXer merges all the content retrieved in an in-memory graph using the *rdflib* library². The relevant triples in this graph are then served to the rest of the modules on demand in an iterative way. Note that this approach prioritizes to reduce execution times over saving memory, as it avoids the repetition of queries to a remote endpoint by creating a local temporal copy of the relevant content. This means that it is discouraged to process too large parts of RDF graphs by consuming its endpoint, as it may lead to memory errors. The recommended approach to process Big Data is processing local dumps.

4.2.2 Instance tracker

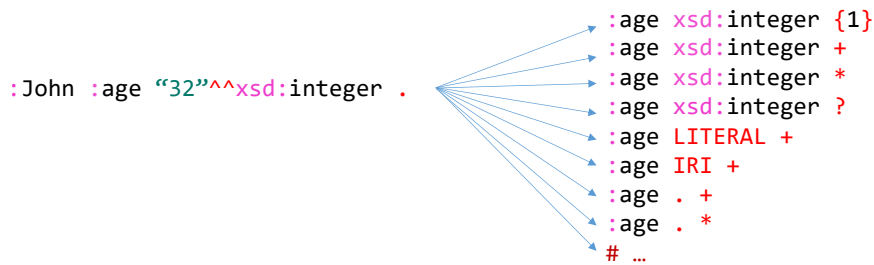
The mission of the Instance Tracker (IT) is determining which nodes (aka instances) will be used to build which shapes. The nature of this process can vary depending on the type of input and the target shapes and instances.

For example, sheXer allows for using shape maps to link a shape label with some entities. If the shape map contains a SPARQL query, this query should be executed to locate those entities. A similar situation occurs when the input is provided via a SPARQL endpoint. In contrast, when the target shapes are specified using a list of target classes, the process consists of finding the instances of those classes. These instances can be located by processing the graph iteratively, with no need of executing an actual SPARQL query. sheXer also lets the user to generate a shape for every class with at least one instance in the target graph. Our proposal can combine some of those strategies too. For example, one can request shapes with custom instance-class associations via shape maps and, at a time, shapes for every class in the target KG.

The IT outputs a dictionary that links instances (keys) with their list of target shapes (values). Note that a given node can be used to build different shapes in

²<https://rdflib.readthedocs.io/en/stable/> Accessed in 2022/05/03.

FIGURE 4.4: Example of constraints that could get positive votes from a certain triple.



case it is instance of several target classes or it appears in the result of different node selectors in a shape map.

4.2.3 Feature Tracker

The Feature Tracker (FT) finds a candidate set of features for each target shape using a voting system. The FT receives the instance dictionary produced by the IT. Then, it processes the triples sent by the GI. The triples are used to cast positive votes for some constraints. sheXer annotates the instance dictionary with the number of times that a combination of predicate and object type is found for each instance. These constraints can have different specificity w.r.t. predicate and object. In Figure 4.4³, we show an example of several candidate constraints supported by the triple $t_e = (:John :age "32"^^xsd:string)$.

Potentially, each triple could cast positive votes for infinite combinations of type of object and cardinality. Indeed, just cardinality could produce infinite options, as every range which includes one occurrence of t_e ($\{0,2\}$, $\{1,2\}$, $\{0,3\}$...) could receive a positive vote. Our current implementation of sheXer limits the positive votes to some representative object and cardinality combinations using the following criteria:

- Exact cardinality, i.e., exact number of triples where a combination of property and object type occurs in the dataset for a given entity.
- The range $\{1, \text{unbounded}\}$, represented by the positive closure ‘+’.

sheXer can produce shapes whose range includes 0 occurrences of a given constraint. However, they are generated in the Shape Adapter module. Regarding the object’s specificity, the following constraints get positive votes:

- **When the object is a literal:** the exact type of the literal, the macro `LITERAL` (any literal), and the macro ‘.’ (any element).
- **When the object is an URI:** the macro `URI` (any URI) and the macro ‘.’. In case the URI is used to extract a shape s , also the label of s .
- **When the object is a blank node:** the macro `BNODE` (any blank node) and the macro ‘.’.

The maximum number of votes that a constraint of a shape s can obtain is the number of instances used to extract s . Once all nodes have been explored, each constraint c_s is associated with a trustworthiness score $\theta_{c_s} = \frac{n_{c_s}}{n_s}$, where n_{c_s} is the number of positive votes to c_s and n_s is the number of instances of s .

³The constraints shown in this Figure are written for shapes in ShExC.

The user can specify a minimum $\theta_U \in [0, 1]$. When the FT finishes, every shape s is associated with a candidate set of constraints which is at least supported by $n_s \cdot \theta_U$ of its instances. This means that every constraint c such that $\theta_{c_s} < \theta_U$ is discarded from the results.

4.2.4 Shape Adapter

The Shape Adapter (SA) analyzes the shapes outputted by the FT to filter, modify, and merge some constraints. Finally, it sorts them. For such a task, the SA uses Algorithm 2. Several symbol conventions must be clarified for a proper understanding of that algorithm:

- Every function or macro used to encapsulate any behavior is denoted as $f_a(x)$, where a is an identifier.
- S is the set containing all the target shapes.
- We denote the constraints associated to a shape s with C_s .
- $f_U(X)$ receives a set of constraints X and returns a collection of sets U . Each $T_U \in U$ is a group of constraints that have the same property and type of object, but different cardinality.
- $f_{d_U}(X)$ receives a set of constraints X which are expected to have the same property and type of object, and it returns the *dominant* constraint α_U of the

Algorithm 2 Shape Adapter: filtering stage pseudo-code

Input: $S = \text{target shapes}$

```

1: for each  $\{s \mid s \in S\}$  do
2:    $C'_s \leftarrow \emptyset$ 
3:    $U \leftarrow f_U(C_s)$ 
4:   for each  $\{T_U \mid T_U \in U\}$  do
5:      $\alpha_U \leftarrow f_{d_U}(T_U)$ 
6:      $I_{\alpha_U} \leftarrow \emptyset$ 
7:     for each  $\{c \mid c \in T_U \wedge c \neq \alpha_U\}$  do
8:        $I_{\alpha_U} \leftarrow I_{\alpha_U} \cup f_{\#}(c)$ 
9:      $C'_s \leftarrow C'_s \cup \{\alpha_U\}$ 
10:   $C''_s \leftarrow \emptyset$ 
11:   $V \leftarrow f_V(C'_s)$ 
12:  for each  $\{T_V \mid T_V \in V\}$  do
13:     $\alpha_V \leftarrow f_{d_V}(T_V)$ 
14:    if  $\nexists I_{\alpha_V}$  then
15:       $I_{\alpha_V} \leftarrow \emptyset$ 
16:    for each  $\{c \mid c \in T_V \wedge c \neq \alpha_V\}$  do
17:       $I_{\alpha_V} \leftarrow I_{\alpha_V} \cup f_{\#}(c)$ 
18:     $C''_s \leftarrow C''_s \cup \{\alpha_V\}$ 
19:   $C_s \leftarrow C''_s$ 

```

▷ Stage 1:

▷ Stage 2:

▷ Stage 3:

set. By default, α_U is the constraint with the highest trustworthiness. In case of tie, the α_U is the one with the most restrictive cardinality. However, the dominance criteria can be configured differently by the user⁴.

- The constraints can have some text comments associated. We denote the comments associated to a constraint c as I_c .
- $f_{\#}(x)$ receives a constraint x and returns a textual comment with some relevant information of x , such as θ_{x_s} , cardinality, and type of object.
- $f_V(X)$ receives a set of constraints X and returns a collections of sets V . Each set in V contains constraints that have the same property but different type of object.
- $f_{d_V}(X)$ receives a set of constraints X and returns a dominant constraint α_V . By default, a constraint $c \in X$ is found the dominant constraint α_V of X when two conditions are met. First, for any $c_i \in X : \theta_{c_i} < \theta_c$, the object type of c subsumes the object type of c_i . Second, there cannot be any $c_i \in X : \theta_{c_i} > \theta_c$ such that the object type of c_i subsumes the type of c . Informally, this means that the object type of c is as specific as possible and, at a time, c is supported by as much instances as possible⁵.

Algorithm 2 can be divided in several consecutive stages.

Stage 1 (lines 2 to 9) - The original set of constraints C_s of s is transformed into a new set C'_s , such that every $c \in C'_s$ has a unique combination of property and object type in C'_s . If C_s already meets this condition, then $C_s = C'_s$. The constraints of C_s not included in C'_s will not appear in the final shape as actual constraints, but its information is kept so it can be presented to the user as text comments. A discarded constraint c_d is transformed into a text comments and associated with a non-discarded constraints having the same property and type of object as c_d .

For example, let us suppose a group of constraints composed by $c_1 = (foaf:name xsd:string +)$, with $\theta_{c_1} = 1$, and $c_2 = (foaf:name xsd:string \{1\})$, with $\theta_{c_2} = 0.9$. This group is related to a shape s . $\theta_{c_1} > \theta_{c_2}$, so c_1 is picked as the dominant constraint of the group. However, the user may find relevant also that 90% of the target instances associated to s has exactly one *foaf:name*. sheXer uses in-line text comments associated with c_1 to provide the information related with c_2 .

Stage 2 (lines 10 to 18) - The constraints with the same property are transformed into a single dominant constraint. Note that the set of constraints transformed in Stage 2 is C'_s (see line 11), so every constraint has already a unique combination of property and object type. The constraints in C'_s may already have some comments associated that should be considered when adding a new comment in this stage.

There are two fundamental algorithmic differences between Stage 1 and Stage 2. The main one is the nature of the function $f_{d_V}(T_V)$. Although it is not its default behavior, this function, in contrast with $f_{d_U}(T_U)$ in line 5, can return a dominant

⁴The criteria to choose dominant constraints can be configured by the user in different ways. For example, more general cardinalities could be chosen as preferment. See sheXer's documentation for further details.

⁵This dominance criteria can be configured as well. For example, constraints with *oneOf* operators could be generated. Also, some tolerance thresholds to keep precise constraints that do not cover all the cases can be defined.

constraint $\alpha_V \mid \alpha_V \notin T_V$. This can happen when the user configures sheXer to return constraints with the operator *oneOf* to merge constraints that have the same property but different type of object.

The second difference is a consequence of the previous one. When the algorithm obtains $\alpha_V \mid \alpha_V \in T_V$, then α_V already has an associated set of comments I_{α_v} (empty or not). In contrast, if $f_{d_U}(T_U)$ creates a dominant constraint $\alpha_V \mid \alpha_V \notin T_V$, then it is necessary to create a new set of comments I_{α_v} associated to α_V .

Stage 3 (line 19) - Finally, C'_s becomes the set of constraints of s . At this point, every constraint associated to s has a unique property in s and can have extra information in text comments.

The SA still needs to perform some iterations which are not described in Algorithm 2 but can modify some constraints or even remove some shapes. Those iterations are described in the following subsections.

4.2.4.1 Extending cardinalities

As already stated, the FT outputs constrains whose cardinality does not include the possibility of zero occurrences, such as *optional* ('?') or *none-to-many* ('*'). This can lead to situations where an instance i used to build a shape s does not conform with s . Let us suppose that sheXer is configured to extract shapes with $\theta_U = 0.8$, and a shape s is obtained. s includes a constraint $c = (foaf:name xsd:string +)$ with $\theta_c = 0.9$. c has been included in s because $\theta_c > \theta_U$, but $\theta_c = 0.9$ means that 10% of the instances does not support c , ergo they do not conform with s .

sheXer includes a configuration option that allows modifying conflictive cardinalities, so every instance conforms with its related shapes. When this option is active, the cardinality of every constraint c whose $\theta_c < 1$ is modified to include a range with zero occurrences. Constraints with cardinality '+' are changed to '*'. Cardinalities of '{1}' are changed to '?'. In general, every cardinality is modified to include the zero-case without losing precision w.r.t. its maximum number of occurrences.

The original cardinality and its θ_s are associated with the constraint as a textual comment. With this, the results include the proportion of instances that conform with the constraint excluding the zero-case.

When the zero-case is included, the θ_c of any constraint c automatically raises to 1. When using Kleene closure, this is trivial to justify, since any instance can match any constraint with a cardinality zero-to-many. When using cardinalities more restrictive with the maximum value, the trustworthiness is always 1 anyway. The fact that there is a constraint c with a restriction for the maximum value in the current stage can only happen when all the instances under analysis have at most that max value of occurrences for a given feature. The new range cardinality with the zero-case is included to conform also with those instances which does not conform with the minimum value.

4.2.4.2 Sorting triple constraints

The shapes extracted can include a large number of constraints, and the user may not want to read or consider them all.

The SA sorts a shape's constraints in decreasing order w.r.t. their θ_c . With this, the most reliable constraints are shown first. Constraints including the zero-case are sorted using the θ_{c_s} computed before the zero-case was included. Otherwise, any constraint including the zero-case would be placed at the top of its shape, since its

score is always one. This could raise a wrong idea of the actual trustworthiness of a given constraint. Scores with the zero-case are modified just to ensure compilation with every instance, but the relations described in such constraints may be infrequent among the seed instances. In contrast, the relations of constraints with no zero-case and a score of 1 are always observed in the neighborhood of every seed instance.

4.2.4.3 Removing empty shapes

Sometimes, sheXer can produce shapes which do not contain a single constraint. This can happen due to two reasons:

- No instance was found to build a certain shape, so sheXer could not infer any constraint.
- Every extracted constraint c_S of a shape S is discarded before the execution of the SA module because its score θ_{c_S} is lower than the threshold θ_U specified by the user.

sheXer can remove those shapes from the results using the following approach:

1. It locates those shapes that do not have any associated constraints and remove them. It annotates the label of those shapes.
2. It explores each constraint that remain in the results to erase references to the shapes removed in the previous iteration. In general, this operation consists in transforming specific shape labels into **IRI** macros.

Our library allows the user to configure whether empty shapes should be removed from the results.

4.2.5 Shape Serializer

The Shape Serializer (SS) transforms the in-memory information into an actual output for the user. The complexity of this task depends on the divergence between the target output format and the conceptual information outputted by the SE. Our library includes two different implementations of the SS: one for the generation of ShEx (in ShExC format) and another one for the generation of SHACL (in turtle format).

Both implementations of the SS are trivial. Even if ShEx and SHACL are not fully compatible [42], the subset of shape features used by sheXer can be represented in both languages.

4.2.6 Computational complexity analysis

The computational complexity of sheXer is the sum of the complexities of its modules, which are sequentially executed. The base complexity of the SA and SS modules is $O(c^2/s)$ and $O(c)$ respectively, where c is the total number of constraints extracted and is s the number of target shapes. The SA's $O(c^2/s)$ complexity is an approximation supposing a balanced number of constraints per shape and it comes from Algorithm 2. Each constraint is compared with the rest of its shape's constraints, so unbalanced constraint distributions could lead to higher complexities. The worst case, where a single shape has all the constraints, will be executed in $O(c^2)$. The rest of the SA's stages are executed in $O(c)$.

The complexity of the IT and the FT modules is tightly linked with the nature of the input. For example, the IT can be executed trivially in $O(1)$ when the instance-shape relation is provided as part of the input. However, when a RDF file needs to be parsed for this task, the process takes $O(t)$, being t the number of triples. The complexity can be higher if a SPARQL endpoint is involved in the process. The FT behaves similarly, as both the IT and the FT depend on the GI's execution.

In section 4.3, we perform several experiments to extract shapes from local RDF files. Under these conditions, the IT and the FT have the following base complexity:

- **IT:** $O(t_c)$, where t_c is the number of instance-class triples.
- **FT:** $O(t + t_i)$, where t is the number of triples and t_i is the number of triples whose subject is a relevant instance.

With this, sheXer would be executed in $O(t_c) + O(t + t_i) + O(c^2/s) + O(c) = O(t + t_c + t_i + c^2/s)$. Note that $t \geq t_c$ and $t \geq t_i$. Note also that the only non-linear complexity is $O(c^2/s)$. However, when computing large datasets where $t_i \gg c$, this part of the algorithm is not the most expensive. Many instances usually generate a relatively small number of constraints. Then, it takes more time to generate constraints from those instances in $O(t_i)$ than to process later those few constraints in $O(c^2/s)$.

4.3 Experiments

Shape languages are relatively new, and so is the problem of automatic shape extraction. To the best of our knowledge, there is not yet a published benchmark to compare the correctness nor performance of existing approaches. The experiments of automatic extractors already published range from mostly qualitative comparisons to performance or scalability analyses.

In this chapter, we perform experiments to check the performance of sheXer in two dimensions: memory consumption and execution time. We have executed sheXer to extract shapes from three well known LD data sources: Wikidata⁶, YAGO⁷, and DBpedia⁸. Details about these computations can be found in Table 4.1.

In our experiments, we always parse local RDF content. Alternative ways of input, such as querying an endpoint, would depend on the endpoint's performance and availability. This could introduce factors that may not be related to sheXer's actual performance in the experiments.

Processing local RDF content is the recommended approach to handle big data sources. With such an approach, sheXer can use parsers that do not keep in memory a full representation of the target graph, but they are able to read the input data in an iterative way and keep in memory just those pieces of information relevant for the shape extraction process.

We run our experiments in a virtual machine with the following specifications: Debian 8 *Jessie* OS, Intel Xeon E5502 processor 1.87 GHz, 32GB RAM, HDD disk

⁶Only triples using Wikidata direct properties in the namespace <http://www.wikidata.org/prop/direct/> where used to extract shapes. The source can be downloaded at <https://archive.org/details/wikidata-json-20150518> Accessed in 2022/05/03.

⁷We computed YAGO3, which can be downloaded at <https://yago-knowledge.org/downloads/yago-3> Accessed in 2022/05/03.

⁸We used a subgraph containing mapping-based literals and objects, as well as class-instance relations. This collection can be downloaded at https://databus.dbpedia.org/danifdezalvarez/collections/latest_mapping_sheXer_test Accessed in 2022/05/03.

Dataset	Target shapes	Dataset size (GB)	N° of triples (millions)	N° of instances (millions)	Memory usage (GB)	Execution time (hours)
Wikidata dump 2015-05-18	1000 (top 1000 classes with more instances)	42.0	991.6 M	13.0 M	25.2	38.3
YAGO3	1000 (top 1000 classes with more instances)	10.3	138.3 M	5.3 M	12.6	17.2
English chapter of Dbpedia	422 (every class in the DBpedia Ontology with instances)	6.0	44.0 M	6.6 M	16.1	4.1

TABLE 4.1: Basic information about the YAGO, Wikidata and DBpedia computations.

with a read speed of 145 MB/s measured with the `hdparm`⁹ command. We set an arbitrarily low threshold $\theta_U = 0.01$ to discard noisy marginal features for every computation. $\theta_U = 0.01$ discards constraints that comply with less than 1% of the total instances considered. The Python version used to run sheXer has been 3.6.4.

As one can see, the time and memory consumption to get a result are different for each source. Several parameters of the input affect the numbers obtained. In the following subsections, we propose scenarios with different inputs to analyze the impact of each parameter on our proposal’s performance.

The actual content of the triples does not have any effect on our approach’s performance. The feature inputs under study are dataset size, number of instances, and number of shapes to extract. These parameters will be arbitrarily altered using a single data source (DBpedia) for every experiment. Memory usage and time consumption are measured at each test scenario. This let us trace performance curves.

In Figure 4.5, we show a fragment of one of the shapes extracted from DBpedia. As one can see, there is just one triple constraint with no zero-case that complies with every instance. Such triple constraint indicates that a node conforming with the shape `:TelevisionShow` must be of type `dbo:TelevisionShow`. Excluding the zero-case, the compliance of the rest of the constraints ranges from 77.8% to 59.2%.

Most of those constraints would have a cardinality of ‘+’ without the zero-case. However, some few nodes cause these constraints to not have a cardinality of exactly ‘{1}’, as the score difference between constraints with ‘+’ and ‘{1}’ cardinality is always relatively low. Note in the comments that the property `dbo:network` is frequently used to point to nodes conforming with the shapes `:TelevisionStation` (25.6% of the instances) and `:BroadcastNetwork` (16.6% of the times). However, many other times, this property is used to point nodes that are not typed with any target class, so sheXer uses the macro `IRI` instead of any specific shape label for the dominant constraint with `dbo:network`.

The shape shown in Figure 4.5 has been shortened in the interest of brevity, as its purpose in this document is to let us discuss a sheXer example output. The complete version of the `:TelevisionShow` shape extracted is available on our published results¹⁰.

⁹<https://linux.die.net/man/8/hdparm> Accessed in 2022/05/03.

¹⁰http://data.weso.es/sheXer/dbpedia/all_shapes.shex Accessed in 2022/05/03.

FIGURE 4.5: Fragment of the DBpedia’s shape :TelevisionShow in ShExC

```

1
2 :TelevisionShow
3 {
4   rdf:type [dbo:TelevisionShow] ; # 100.0 %
5   dbo:runtime xsd:double *;
6   # 77.80100848604108 % obj: xsd:double. Cardinality: +
7   # 74.30820317304145 % obj: xsd:double. Cardinality: {1}
8   dbo:releaseDate xsd:date *;
9   # 74.45988603287829 % obj: xsd:date. Cardinality: +
10  # 72.85901693108679 % obj: xsd:date. Cardinality: {1}
11  dbo:numberOfEpisodes xsd:nonNegativeInteger *;
12  # 67.68335178124872 % obj: xsd:nonNegativeInteger. Cardinality: +
13  # 66.30180789570778 % obj: xsd:nonNegativeInteger. Cardinality:
14  {1}
15  dbo:network IRI *;
16  # 59.207559545771325 % obj: IRI. Cardinality: +
17  # 52.248595908662324 % obj: IRI. Cardinality: {1}
18  # 25.64260238593039 % obj: :TelevisionStation. Cardinality:+
19  # 16.60312384700529 % obj: :BroadcastNetwork. Cardinality: +
20  # ...
21 }

```

4.3.1 Limiting the number of instances used

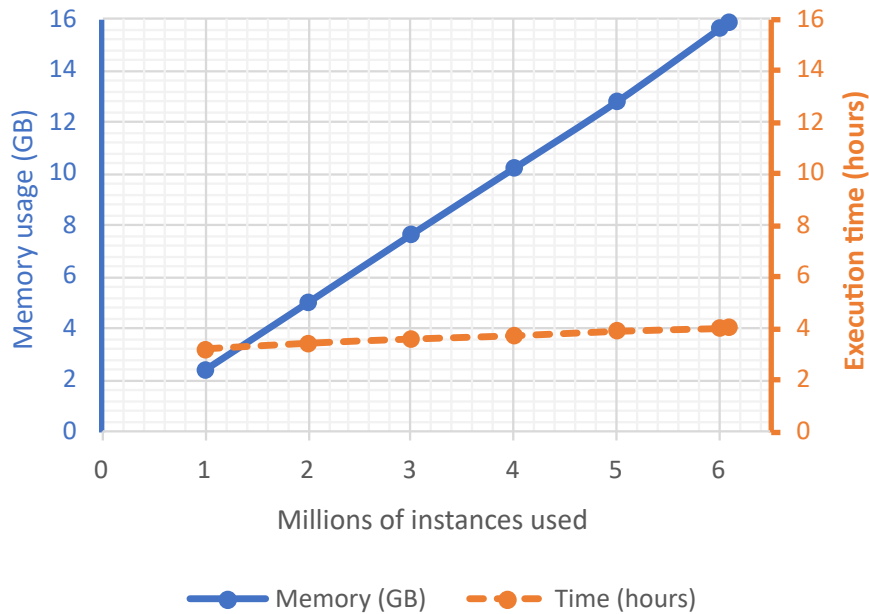
As shown in Table 4.1, the number of instantiation triples in DBpedia is higher than 6.6 million. We repeated the process of shape extraction limiting the number of instantiation triples to a certain number. We started in 1M triples and repeated the computation with arbitrary increments of 1M instances each time, until every instance of every target class was used. The results regarding execution time and memory consumption are shown in Figure 4.6.

As one can see, the number of instances has a linear relation with execution time and memory consumption. The impact on memory consumption is caused by the instance dictionary generated by the IT and the FT. The more instances, the bigger becomes this dictionary and its associated memory usage. The effect on execution time is lower than the impact on memory usage because sheXer is parsing the whole content in every case. This parsing process sets a minimum execution time, which is more significant in environments where the I/O disk speed is relatively low. The FT module evaluates whether a triple sent from the GI is relevant. The relevant triples trigger extra computations in the TF and SA modules. The more instances are considered, the more triples become relevant. This causes the linear relation between the number of instances and execution time.

4.3.2 Limiting the number of target shapes

As stated in section 4.2.6, sheXer’s complexity depends, among other parameters, on the number of constraints produced. However, the number of constraints cannot be known a priori. In opposition, in case there is a balanced distribution of the number of constraints per shape, the number of shapes, which can be known a priori, can be used to estimate execution times and memory usage. In this subsection, we study the impact on the performance of the number of target shapes.

FIGURE 4.6: Performance of sheXer with different amounts of instances used.



In our experiments, we started with just 20 target shapes and then perform arbitrary increases of 20 shapes for each iteration until every class with at least one instance was used. As already shown in section 4.3.1, the number of instances has a crucial impact on the performance. Then, the more instances a class has, the greater is its impact. To avoid erratic numbers due to classes with an unbalanced number of instances, we used an arbitrarily small number of instances to be considered (as most) per each class. We picked this limit so 90% (380 out of 422) of the DBpedia classes has at least this number of instances. In our dataset, *dbo:Chancellor* ranks 380th w.r.t. to number of instances, with a total of 57. Then, the limit picked was 57 instances. The results obtained are shown in Figure 4.7. Note that the memory scale in the y-axis is different from the rest of the figures. It ranges from 0 to 200 MB instead of 0 to 16 GB.

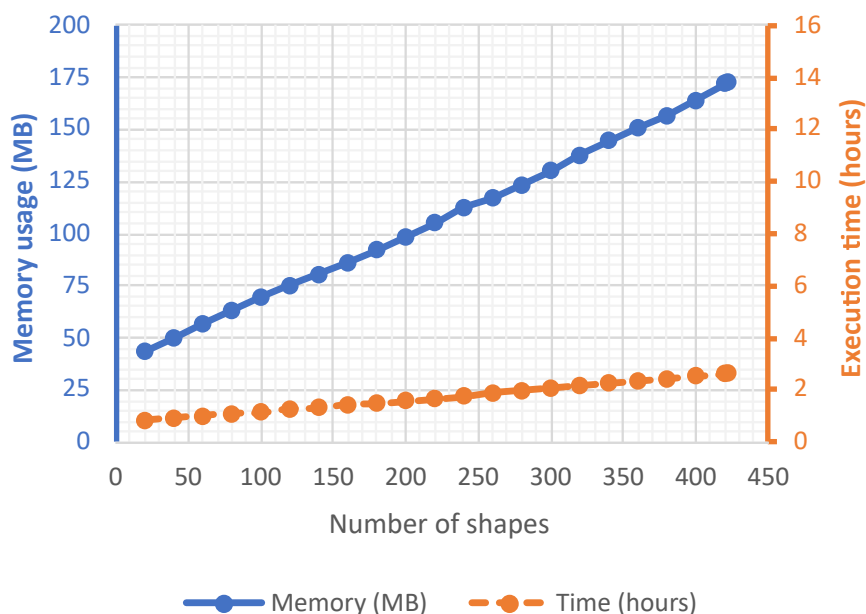
As one can see, using a relatively small number of instances drastically decreases memory usage. However, there is a linear relationship between the number of shapes used and memory consumption. This relation is explained by two factors: on the one hand, the FT generates an abstract profile in main memory for each shape. On the other hand, each shape causes a growth in the instance dictionary proportional to its number of instances.

There is also a linear relation with execution time, with a trend similar to the one observed in Figure 4.6. However, there is a noticeable difference in execution time between Figures 4.7 and 4.6 at the maximum values of the x-axis, explained by the limit of 57 instances per class used in Figure 4.7. The difference of 1.39 hours is the time used to compute the instances discarded in Figure 4.7's experiment.

4.3.3 Limiting the amount of triples

In this subsection, we study the effect of the number of triples processed on sheXer's performance. Since the impact of the number of instances has already been studied, in this experiment, we kept the same number of instantiation triples across all iterations. Every instance was used in every case. We did change the number of

FIGURE 4.7: sheXer's with different number of target shapes.



entity-to-entity and entity-to-literal triples. Our DBpedia dataset contains 37.328M of these two kinds of triples.

We performed iterations starting at the arbitrary amount of 5M triples (without counting the instantiation triples). Then, we made an arbitrary increment of 5M triples for each iteration until reaching the total 37.328M triples. The number of triples of each kind added at each iteration is proportional to the total number of triples available. In the first iteration, we used 2.698M object triples and 2.302M literal triples, which make together 5M elements. We add the very same number of triples of each type at each iteration. The results are shown in Figure 4.8.

As one can see, memory usage stays stable and independent of the number of triples used. There is not a determinant linear relation between memory usage and triples which are not expressing a class-instance relationship.

However, there is a linear relation with execution time caused by the different number of triples relevant for the process in each iteration. Those triples are potentially spread across the whole dataset. Then, the bigger is the slice of the dataset used, the higher are the chances to find this kind of triples.

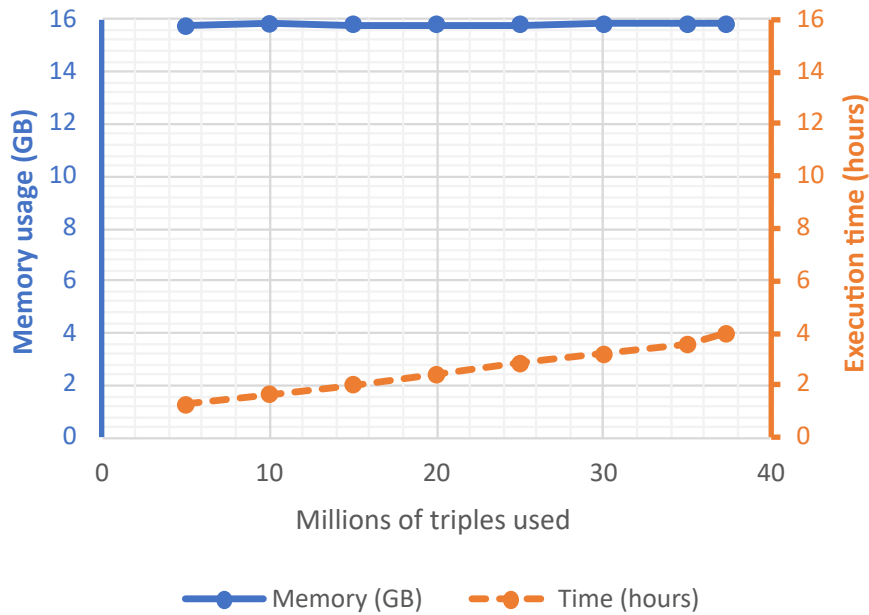
4.3.4 Convergence w.r.t. number of instances used

As already shown, sheXer's performance scales linearly w.r.t. some features of the input dataset. Too ambitious goals can lead to high rates of memory usage or execution time.

The structure with more impact on memory usage is the instance dictionary generated by the IT. The instance dictionary is a key element to produce shape interlinkage in reasonable execution time, as the operation to check the class of a given instance is frequent, and it allows to perform it in $O(1)$ complexity. It is also used to produce precise cardinalities.

To cope with this limitation, we explore whether using a relatively small number of instances per shape can be enough to extract accurate shapes. We extracted shapes for every DBpedia class in different iterations, using an increasing maximum

FIGURE 4.8: Performance of sheXer with different dataset sizes.



number of instances per shape at each iteration. We started using two random instances per class. Then, we performed successive iterations doubling the maximum number of instances. In figure 4.9, we show the changes detected between consecutive iterations. Note that the scale of the x-axis in this chart is logarithmic. At the end of each iteration, we checked two factors:

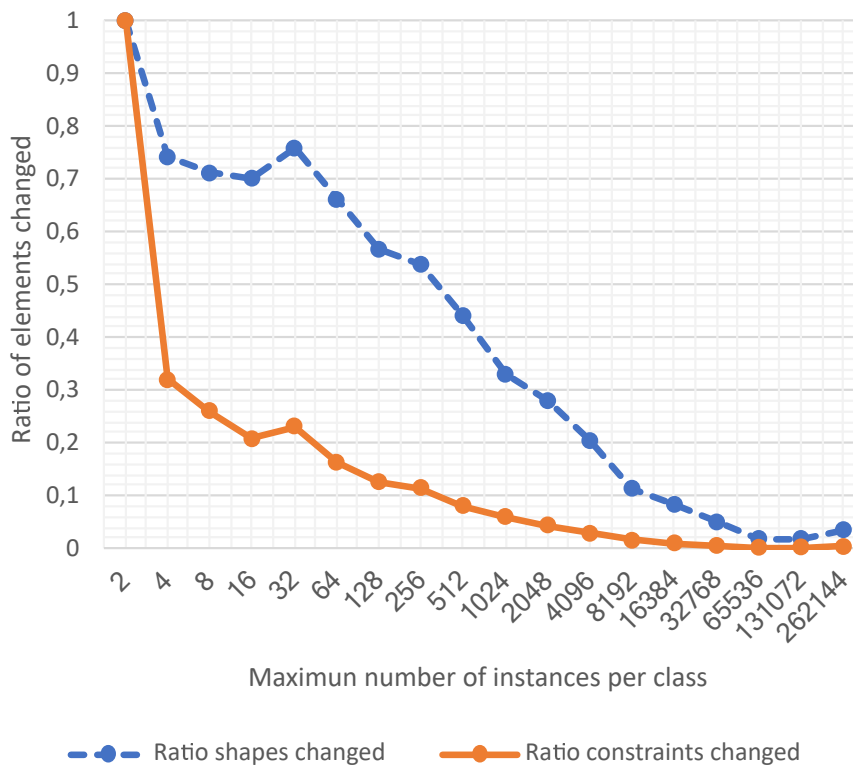
1. The number of total changes w.r.t. the previous iteration. We count as a change in a shape one of the following events:
 - Gaining a triple constraint.
 - Loosing a triple constraint.
 - Having a modification in any element of a triple constraint.
2. The number of shapes changed w.r.t. the previous iteration. We consider that a shape has changed when there is at least one change among its triple constraints.

The numbers shown in the y-axis are relative. For shapes, we show the proportion of elements that changed w.r.t. the total number of shapes, which is 422 in every case. For triple constraints, we show the proportion of changes detected w.r.t. to the total number of triple constraints at each iteration.

As one can see, with very few exceptions, every iteration causes fewer changes than its previous iterations. With a high enough instance limit, the shapes tend to converge. For example, using 8192 instances per class, just 1,5% of the constraints are changed. These changes affect 11% of the shapes. For any other iteration with a higher instance limit, the proportion of shapes getting some change is always lower than 10% and affects 1% or less of the constraints.

The ratios of shape and constraint changes detected for every iteration are also available in Table 4.2, as well as some other information related to the shapes' evolution. Let C be the set of all target shapes to extract. An increment of x units in the instance limit for a given iteration would introduce $|C| \cdot x$ new instances on the

FIGURE 4.9: Shape convergence using different amounts of instances per shape.



computations just in case every $c \in C$ has at least x instances still not considered. Since this is not always the case, the actual number of new instances introduced in the experiments in successive iterations is shown in the sixth column of Table 4.2.

The number of new instances for each iteration allows us to calculate what we called Shape-Instance Performance (SIP) and Constraint-Instance Performance (CIP). SIP is shown in the seventh column of Table 4.2, and it is defined as the relation between the number of instances used and the number of shapes changed. SIP can also be interpreted as the number of instances required to cause a single shape change. As one can see, the more instances are processed, the smaller is the effect of a single instance over the results, and the SIP grows rapidly with each iteration. For example, at the already mentioned limit of 8192 instances per class, 9267 new instances are worth a single shape change.

CIP is shown in the eighth column of Table 4.2 and it is defined as the relation between the number of instances used and the number of constraints changed. As one can see, the CIP is slightly better than SIP for almost every iteration. However, both numbers tend to stay in the same magnitude order.

Table 4.2 and Figure 4.9 indicate that a representative sample of instances can be enough to extract highly accurate shapes. Using 8192 as instance limit, the total instances computed are close to 1.3M. As shown in Figure 4.6, this leads to a memory usage close to 3.2GB, which is five times smaller than the 16GB needed to compute the dataset using every available instance.

4.3.4.1 Discussion on execution times

The experiments performed reveal that sheXer can be applied over large real-world RDF datasets. The linear relation between memory consumption and instances used

Instance limit per class in previous iteration	Instance limit per class in current iteration	Total constraints	Ratio of shapes that changed	Ratio of constraints that changed	Actual number of new instances used	Shape-Instance Performance (SIP)	Constraint-Instance Performance (CIP)
0	2	2842	1,000	1,000	840	2,0	0,30
2	4	3409	0,742	0,319	827	2,6	0,76
4	8	3992	0,711	0,261	1637	5,5	1,57
8	16	4550	0,701	0,208	3227	10,9	3,41
16	32	4303	0,758	0,232	6342	19,8	6,36
32	64	4214	0,661	0,163	12239	43,9	17,84
64	128	4292	0,566	0,127	23547	98,5	43,36
128	256	4378	0,538	0,115	44335	195,3	88,14
256	512	4392	0,441	0,081	78907	424,2	221,65
512	1024	4409	0,329	0,060	131876	948,7	495,77
1024	2048	4414	0,280	0,044	217935	1846,9	1117,62
2048	4096	4424	0,204	0,028	325744	3787,7	2626,97
4096	8192	4445	0,114	0,015	444817	9267,0	6541,43
8192	16384	4436	0,083	0,010	567820	16223,4	12343,91
16384	32768	4443	0,050	0,005	640852	30516,8	26702,17
32768	65536	4444	0,019	0,002	668062	83507,8	66806,20
65536	131072	4447	0,019	0,002	807333	100916,6	73393,91
131072	262144	4448	0,036	0,003	801842	53456,1	53456,13

TABLE 4.2: Shape convergence with different instance limits per class.

can be an issue for ambitious computations. However, in section 4.3.4, we have shown how the results of sheXer tend to converge when using a relatively small number of instances, tackling the issue of memory usage.

There can also be implementation alternatives that reduce memory usage without causing a critical increment of execution times, such as solutions combining efficient databases with in-memory caches. However, this option has not been yet implemented and cannot be evaluated.

The experiments have revealed a linear relation between execution time and dataset sizes. Even if sheXer is able to compute real-world dataset in affordable time, this relation could be also a scalability issue with larger datasets. To tackle this, the different components of sheXer could be parallelized using usual solutions such as MapReduce[263] in different stages:

- **Instance Tracker.** Every triple is computed in an independent way looking for instances of target shapes. Thus, different nodes could process chunks of the dataset and merge later their results to produce a single instance dictionary.
- **Feature Tracker.** Again, every triple in the dataset is computed independently, in this case to cast positive votes for shape constraints. This stage requires the instance dictionary as input. Different nodes could compute chunks of the dataset sharing an access to a single instance dictionary, or even having

FIGURE 4.10: Python code - Basic example of sheXer usage

```
1
2 from shexer.shaper import Shaper
3
4 local_graph = "/disk/path/to/local/ntriples_file.nt"
5
6 shaper = Shaper(all_classes_mode=True,
7                 graph_file_input=local_graph)
8
9 result = shaper.shex_graph(string_output=True)
10 print(result)
11
```

its own copy of the dictionary. There would be no need to synchronize the different copies, as the feature tracker use the instance dictionary in read-only mode. The votes generated by each node can be trivially merged later in a single structure.

- **Shape Adapter and Shape Serializer.** These two stages do not process the dataset, but intermediate products of each respective previous stage. These products can be split in every case in shapes that will be processed independently, which allows for implementing parallel computing strategies.

4.4 sheXer working modes

In this section, we expose different settings for sheXer and explain how those settings can be used to tackle different use cases. Also, we provide code snippets to let the reader test the described features using our Python library.

Regardless of the configuration chosen, the way to use our library is quite straightforward. One must instantiate a `Shaper` object with some initial parameters, and then call the method `shex_graph()` on the newly created object. A toy example of this is shown in Figure 4.10. This code prints in the standard output the shapes extracted from a file located in the disk path indicated with `local_graph`.

4.4.1 Input types

4.4.1.1 Formats

sheXer can process content written in most of the usual RDF syntaxes. The library supports n-triples, turtle, rdf/xml, n3 and json-ld formats. If no input format is specified, sheXer will assume that the content to parse is written in n-triples. To change this, one must give a value to the `input_format` parameter when building a Shaper object. This parameter expects a string indicating the expected syntax, and the accepted values can be found in the package `shexer.const`. An example is shown in Figure 4.11.

sheXer also supports a non-standard input format. If `shexer.const.TSV_SP0` is chosen as input format, sheXer will expect to process content in which there is a triple per line, and each line consist of three elements representing subject, property, and object separated by `tab` (`'\t'`) characters.

It is important to remark that the current implementation of sheXer does not offer the iterative parsing strategy described in Section 4.2.1 for every input format. Some

FIGURE 4.11: Python code - Change input format

```

1
2 # The input_format param could be filled with one of the
3 # following values available in shexer.const:
4 # NT, TSV_SPO, N3, TURTLE, TURTLE_ITER, RDF_XML, JSON_LD
5 from shexer.shaper import Shaper
6 from shexer.const import TURTLE
7
8 shaper = Shaper(all_classes_mode=True,
9                 input_format=TURTLE
10                graph_file_input="/path/turtle_file.ttl")
11
12 print(shaper.shex_graph(string_output=True))
13

```

of the parsers use the `rdflib` library to load the target content, which requires to load the entire target graph in main memory. The formats that allow to process the graph using iterative strategies are the following ones:

- `const.NT` (default option). It processes n-triples.
- `const.TSV_SPO`.
- `const.TURTLE_ITER`. It processes turtle. Note that there are two available options to process turtle graphs. When the user chooses `const.TURTLE`, sheXer uses a parser based on `rdflib`, which is robust but cannot process too large graphs. In opposition, the parser used when the user chooses `const.TURTLE_ITER` allows to save memory but has some known issues when the content makes usage of the syntax “[]” to define blank nodes.

4.4.1.2 Input types

Currently, sheXer provides five different ways to provide input data. In every case, the input type is specified by setting the value of some parameter(s) when creating an instance of a `Shaper` object. Note that these options are exclusive, i.e., a single input type should be chosen in every case.

- **Raw strings.** The content to parse is provided using an in-memory `str` or `unicode` object¹¹. The user should provide this object using the `raw_graph` parameter.
- **Remote files.** sheXer can retrieve and parse content available on-line. This content is specified by providing its URL. To choose this mode, two parameters can be set. In case the whole target content can be retrieved from a single URL, `url_graph_input` should be set with the value of this URL. In case the target content is composed by several files, having each one its own URL, then the user can build a Python list and pass it to the parameter `list_of_url_input`. sheXer will download and process those files as if they were a single graph. Note that, in case of providing a list of values, the content of all files is expected to have the same syntax, which should had been specified using the parameter `input_format`.

¹¹`str` and `unicode` are the Python built-in types to represent raw string and encoded string content respectively.

- **Local files.** This option is analogous to the previous one but, in this case, sheXer expects to find this content in local files instead of remote ones. The parameter `graph_file_input` should be set to provide the disk path of a single file, while the parameter `graph_list_of_files_input` is handy for providing a list of files.
- **SPARQL endpoints.** sheXer can explore an endpoint to retrieve target content by executing different SPARQL queries. For this, the user must provide the URL of the SPARQL endpoint using the parameter `url_endpoint`. Note that, when `url_endpoint` is used, the `input_format` parameter has no effect, as sheXer do no need to parse any file but to process the results of several SPARQL queries.
- **Rdflib graphs.** sheXer can mine `rdflib.Graph` objects with content already loaded. This allows for easily using sheXer with many other Python libraries that work with this library. In such a case, the user should set the parameter `rdflib_graph`.

4.4.1.3 Independent class-instance file

As already stated, sheXer performs two iterations to process the content. In the first one, it locates which instances are used to build which shapes. In the second one, it annotates the features occurring in the neighborhood of those instances. However, when working with graph dumps, there could be no need to explore the whole target content to perform the first iteration. Some sources, such as DBpedia¹², organize their data dumps by splitting the content in several files. Each file contains triples of a certain topic or category. With this, a user interested in just one of those topics/categories does not need to download nor compute the whole dump.

In case the instance-class relations are in a separate file, then sheXer can be configured so just this file is processed during the execution of the IT module. An example of how to do this is shown in Figure 4.12. The path to the class-instance file should be indicated in the parameter `instances_file_input`, and that file will be the only content that the GI will send to the IT. The GI will send to the FT the whole target content instead, which will be usually specified as a list of files using the parameter `graph_list_of_files_input`. Note that the class-instance file should be included in this list too in case the user wants it to be processed by the FT module.

4.4.1.4 Working with endpoints

When sheXer consumes the content of an endpoint, it first locates some target seed nodes (the way to detect those seed nodes is explained in section 4.4.2). Then, it retrieves content surrounding those nodes. This process can trigger a big amount of SPARQL queries, causing issues such as high execution times or error responses from the endpoint due to excessive traffic in short time periods.

sheXer lets the user to configure how to tackle this process to keep a balance between the amount of information retrieved and the performance cost. By default, sheXer tries to retrieve just those triples in which the seed nodes are used as subjects. This can be changed using three parameters:

¹²DBpedia typings can be downloaded in a single file which does not contain any other type of triples. Cf. <https://databus.dbpedia.org/dbpedia/mappings/instance-types/> Accessed in 2022/05/03.

FIGURE 4.12: **Python code** - Standalone file for instance-class relations

```

1
2  from shexer.shaper import Shaper
3
4  shaper = Shaper(all_classes_mode=True,
5                  instances_file_input="/path/class-instance.nt",
6                  graph_list_of_files_input=[
7                      "/path/class-instance.nt",
8                      "/path/some_other_content1.nt",
9                      "/path/some_other_content2.nt"
10                 ])
11
12  print(shaper.shex_graph(string_output=True))
13

```

- `depth_for_building_subgraph`. Default value: 1. It indicates the maximum distance between the seed nodes and the content retrieved. A value of '1' prevents sheXer to create shape-interlinkage except for cases in which there are direct links between seed nodes.
- `track_classes_for_entities_at_last_depth_level`. Default value: `False`. When it is set to `True`, the SPARQL endpoint is explored to find the types of those IRIs found at the depth level indicated in `depth_for_building_subgraph`. That is, sheXer will explore an extra depth level beyond the threshold specified in `depth_for_building_subgraph`, but only to look for typing triples.
- `inverse_paths`. Default value: `False`. When it is set to `True`, sheXer also looks for triples where the seed nodes are objects. If `depth_for_building_subgraph` > 1, then sheXer also looks for triples where the IRIs related to previous seed nodes are used as objects.

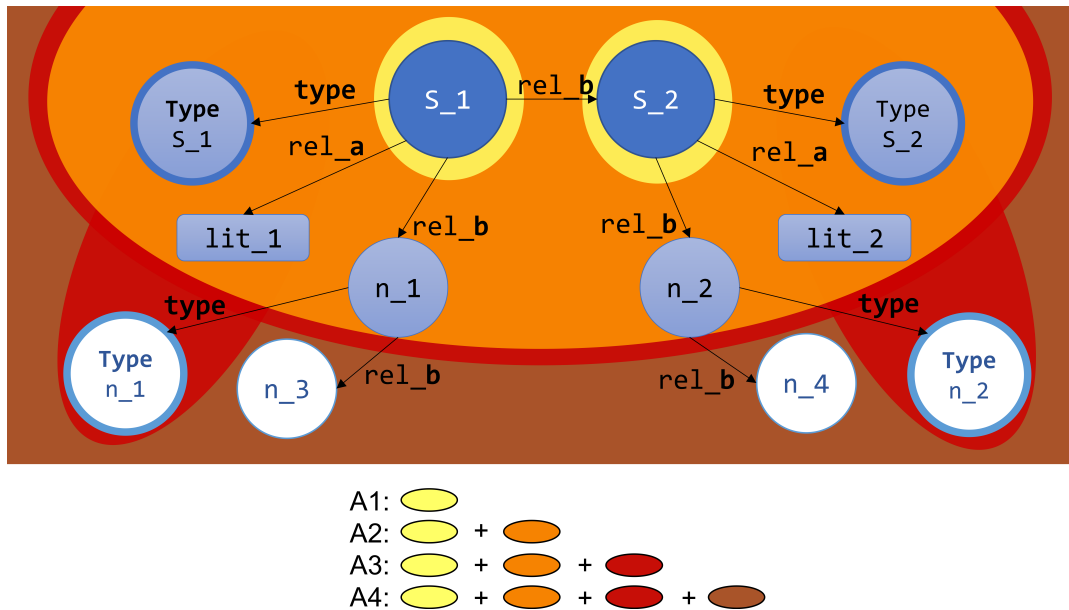
In Figure 4.13, we show an example graph with different areas containing subgraphs to clarify the usage of this parameters. The graph represented in this figure uses the following conventions:

- Elements labeled **S_***value*: IRIs which are seed nodes.
- Elements labeled **n_***value*: IRIs which are not seed nodes.
- Elements labeled **lit_***value*: literal values.
- Relations labeled **type**: links between an instance and its class.
- Relations labeled **rel_***value*: any relation except the one used to link an instance with its class.
- Elements labeled **Type** *value*: class URIs.

Figure 4.13 defines several subgraphs using its background color. The relevant areas to consider are specified and labeled in the Figure's legend. Those areas contain the following elements:

- **A1**. It contains the seed nodes.

FIGURE 4.13: Subgraphs reachable according to different endpoint configurations



- **A2.** It contains every node at distance 1 from the seed nodes. sheXer would retrieve this content with `depth_for_building_subgraph = 1` and `track_classes_for_entities_at_last_depth_level = False`.
- **A3.** It contains nodes at distance 1 and also the classes of those nodes. sheXer would retrieve this content with the setting `depth_for_building_subgraph = 1` and `track_classes_for_entities_at_last_depth_level = True`.
- **A4.** It contains every node at distance 2 from the seed nodes. sheXer would retrieve this content with `depth_for_building_subgraph = 2`. The setting of `track_classes_for_entities_at_last_depth_level` would not have any effect in this case, as the example graph does not include any node at distance more than 2 from the seed nodes.

The code shown in Figure 4.14 includes an example to get a shape for every class consuming a certain endpoint. sheXer will explore the immediate neighborhood of the instances of each class as well as the types of those nodes at distance one from the seed instances.

4.4.2 Target entities and shapes

sheXer needs to be told which nodes should be used to build which shapes. Usually, shapes are linked with classes, so the class's instances are mined to extract a shape for their class. However, sheXer can also extract shapes from custom node groupings using ShEx's shape maps. The current implementation of our library support three main ways to indicate target entities and shapes:

- **All classes mode.** sheXer will produce a shape for every class with at least an instance in the target content. It will use every available instance for each shape. To choose this option, one must set `all_classes_mode = True`.

FIGURE 4.14: **Python code** - Consuming and endpoint with depth=1 + extra classes

```

1
2  from shexer.shaper import Shaper
3
4  # The default value for depth_for_building_subgraph is 1,
5  # so there is no need to set this parameter in case the
6  # user does not want to explore deeper regions of the
7  # graph
8  shaper = Shaper(url_endpoint="https://example.org/sparql",
9                  track_classes_for_entities_at_last_depth_level=True,
10                 all_classes_mode=True)
11
12  print(shaper.shex_graph(string_output=True))
13

```

- **Target classes.** sheXer will produce a shape for each class in a list. The system will use every available instance of those classes to build each shape. The list of target instances can be provided in two ways:
 - Parameter `target_classes`. It expects a Python list containing the URIs of the target classes.
 - Parameter `file_target_classes`. It expects a disk path to a file which is expected to contain a target class URI per line.
- **Shape maps.** sheXer accepts shape maps linking custom node groupings with target shapes. It can parse two syntaxes to define shape maps: the JSON syntax (set `shape_map_format = shexer.const.JSON`) and the specialized shape maps grammar (`shape_map_format = shexer.const.FIXED_SHAPE_MAP`, which is the default value). These two syntaxes are defined in the official shape maps specification¹³. The node selectors of each shape association can be either a single RDF node, or a triple pattern (cf. 2.6.1 for further details about those selectors). Additionally, sheXer accepts SPARQL patterns as node selectors. A SPARQL pattern is a single-column SPARQL query whose results, i.e., the set of nodes retrieved by that query in a given endpoint, are associated with a shape label. SPARQL patterns are not yet included in the official shape map specification, but some ShEx implementations support them¹⁴. Two parameters can be used to choose this option:
 - Parameter `shape_map_raw`. It expects an `unicode` or `str` object containing the shape associations.
 - `shape_map_file`. It expects a disk path to a file containing shape associations (one per line).

4.4.2.1 Finding target entities in endpoints

As stated in section 4.4.1, the process of retrieving content from a SPARQL endpoint starts by selecting some seed nodes. The seed nodes are found by executing some SPARQL queries with the following criteria:

¹³<http://shex.io/shape-map/> Accessed in 2022/05/03.

¹⁴Cf. <https://github.com/shexjs/shex.js> Accessed in 2022/05/03.

- **All classes mode.** Every node which is instance of any class is used as seed node¹⁵.
- **Target classes.** Every node which is instance of a target class is used as seed node. sheXer executes separated SPARQL queries to find the instances of each class.
- **Shape maps.** Each node selector of each shape association in a shape map is mapped to a SPARQL query (except for node selectors which are actual RDF nodes). This SPARQL query should have a single column result. There are two different cases of node selectors which trigger SPARQL queries:
 - **Triple patterns.** Valid triple patterns should contain exactly one *focus selector*. The query generated is the one which identifies the nodes which fits the focus of the triple pattern.
 - **Actual SPARQL selectors.** SPARQL selectors will be executed to find seed nodes as long as they produce single-column results. If this condition is not met, sheXer raises an informative error.

4.4.2.2 Compatibility between modes

When sheXer works with a list of target classes or in `all_classes_mode`, and it is told to produce SHACL content, the shapes are linked to its corresponding class with the property `sh:targetClass`¹⁶. However, note that shape maps are a concept exclusively defined in ShEx. sheXer accepts to be configured to produce SHACL content and use shape maps to determine target entities anyway. Nevertheless, a selection of targets based on shape maps cannot ensure that every individual used to build a shape is an instance of a common class. Then, the resulting shapes will not be linked with a class with `sh:targetClass`.

The `all_classes_mode` is compatible with the other two ways of input specification. In Figure 4.15, we show an example of such a configuration. A shape map is provided, so the node `:Jimmy` is used to build a shape labeled `<Person>`. However, `all_classes_mode` is active too. This means that any class with instances will generate a shape too.

When `target_classes` and `all_classes_mode` are both active, the set of shapes extracted could contain more elements than the ones produced in an execution which does not define target classes.

This situation can happen in two scenarios:

- `all_classes_mode` causes to extract a shape for every class with at least an instance. `target_classes` causes to produce a shape for every class specified, not mattering if it has instances or not. That is, a class included in the list of `target_classes` could produce an empty shape that could not had been generated by only having `all_classes_mode` active.
- sheXer can be told to remove empty shapes form the results (see section 4.4.9 for further details). However, no shape associated with a class included in the `target_classes` list will be removed from the results.

¹⁵The current implementation of sheXer has this feature disabled. When working against endpoints of large graphs, the SPARQL queries produced cause frequent issues such as timeouts with big results or temporal IP bans due to the execution of too many queries.

¹⁶Cf. <https://www.w3.org/TR/shacl/#targetClass> Accessed in 2022/05/03.

FIGURE 4.15: Python code - Shape maps and all classes mode together

```

1
2  from shexer.shaper import Shaper
3
4  shaper = Shaper(url_endpoint="https://example.org/sparql",
5                  depth_for_building_subgraph=2,
6                  shape_map_raw("<http://example.org/Jimmy>@<Person>",
7                               all_classes_mode=True)
8
9  print(shaper.shex_graph(string_output=True))
10

```

4.4.3 Management of instantiation property

The class-instance relation receives a special treatment by sheXer. On the one hand, it is a usual way to locate seed entities for shapes shapes. On the other hand, sheXer's FT module processes instantiation triples in a different way. Instead of casting positive votes for certain object types (some shape label, the macro `IRI`, or the macro `'.'`), it produces positive votes for constraints with a value set containing the actual class URI found in a triple. An example of this was shown in Figure 4.1 in this chapter's introduction. The described `<CountryShape>` includes a constraint with the property `rdf:type` associated to a value set that indicates that the type of a node conforming with this shape should be specifically the URI `:Country`.

In RDF, the standard property to indicate an instance-class relation between two nodes is `rdf:type`. However, some KGs use different properties for the same purpose. An insightful example of this is Wikidata, which uses `'P31 - instance of'` instead of `rdf:type`. To support such use cases, sheXer allow to specify alternative instantiation properties. To do so, one must provide the URI of this property using the parameter `instantiation_property`. If no value is provided for this parameter, then sheXer will use `rdf:type` by default.

4.4.4 Namespaces management

sheXer can be fed with some prefix-namespace pairs using a Python dictionary. An example of this is shown in Figure 4.16. The parameter `namespaces_dict` expects to receive a dictionary where the keys are namespaces and the values are prefixes.

These prefix-namespace pairs can be used to make inputs and outputs easier to read and write. On the one hand, sheXer's output (both in SHACL and ShEx) will declare and use those prefixes whenever it is possible to avoid absolute URIs. On the other hand, the user can use those prefixes to write URIs in many configuration parameters, such as target classes specification, shape maps, and instantiation property.

When parsing RDF content in turtle syntax, sheXer may find some other prefix declarations. These new prefix declarations will also be used to generate readable ShExC/SHACL output. In case of conflict, i.e., in case there is a prefix declaration for a namespace already paired with a prefix in the `namespaces_dict`, the prefix used will be the one explicitly provided by the user in `namespaces_dict`.

FIGURE 4.16: Python code - Feeding sheXer with namespace-prefix pairs

```

1
2 from shexer.shaper import Shaper
3
4 namespaces = {
5     "http://example.org/" : "ex",
6     "http://www.w3.org/1999/02/22-rdf-syntax-ns#": "rdf",
7     "http://www.w3.org/2001/XMLSchema#": "xsd"
8 }
9
10 shaper = Shaper(all_classes_mode=True,
11                namespaces_dict=namespaces,
12                graph_file_input="/path/local_file.nt")
13
14 print(shaper.shex_graph(string_output=True))
15

```

FIGURE 4.17: Python code - Generation of SHACL content

```

1
2 from shexer.shaper import Shaper
3 from shexer.consts import SHACL_TURTLE
4
5 shaper = Shaper(all_classes_mode=True,
6                 graph_file_input="/path/local_file.nt")
7
8 # output_format also accepts the value const.ShExC
9 # const.ShExC is the value used by default
10 print(shaper.shex_graph(string_output=True,
11                         output_format=SHACL_TURTLE))
12

```

4.4.5 Configuring output

sheXer can be told to produce ShEx and SHACL content. As stated in section 2.6, both languages can be expressed in several syntaxes. However, sheXer produces content in just one syntax for each language in a native way. ShEx is expressed in ShExC, and SHACL is expressed in turtle. By default, sheXer generates ShEx content. One must use the parameter `acceptance_threshold` of the function `shex_graph` to obtain SHACL content. In Figure 4.17, we show an example of such configuration.

The extracted shapes can be returned as a `str` object or written to a file. sheXer returns a `str` object if the parameter `string_output` is set to `True`, such as in the example of Figure 4.17. To write the results to a file, the target path should be provided using the parameter `output_file` of the `shex_graph` function.

At least one of `string_output = True` or `output_file = "/a/path.nt"` should be used when calling `shex_graph`. However, they can be used at the same time, so the content is returned as a `str` and serialized in a file too.

4.4.6 Acceptance threshold

The acceptance threshold used to discard infrequent constraints during the execution of the FT module can be set with the parameter `output_format` of the function `shex_graph`. An example of such configuration is provided in Figure 4.18. Note that

FIGURE 4.18: Python code - Setting an acceptance threshold

```

1
2  shaper = Shaper(all_classes_mode=True,
3                  graph_file_input="/path/local_file.nt")
4
5  # The acceptance_threshold should always be a value
6  # in the range [0,1]. the default configuration is 0
7  print(shaper.shex_graph(string_output=True,
8                  acceptance_threshold=0.3))
9

```

FIGURE 4.19: Python code - Enabling inverse paths

```

1
2  from shexer.shaper import Shaper
3
4  shaper = Shaper(all_classes_mode=True,
5                  graph_file_input="/path/local_file.nt",
6                  inverse_paths=True)
7
8  print(shaper.shex_graph(string_output=True))
9

```

this threshold prevents topological features infrequently observed to be part of the actual shapes, but it also removes every information related to those features from the text comments.

4.4.7 Inverse paths

By default, sheXer extracts constraints in which the focus node acts as subject of a triple. However, our library can also extract inverse constraints, i.e., those in which the focus node acts as object of a triple. This can be configured by setting the parameter `inverse_paths = True` as shown in Figure 4.19.

When `inverse_paths` is active, the performance cost of executing sheXer, both in execution time and memory usage, is increased. The size and composition of the instances dictionary produced by the IT module remains exactly the same. However, the FT module and the results that it produces are affected in two ways:

- More triples could be processed. With `inverse_paths = False`, the FT just process those triples whose subject is among the target instances. However, when `inverse_paths = True`, triples whose object is among the seed instances are processed too, which triggers extra computations (and increase the execution time).
- The instances dictionary produced by the IT is decorated with more feature observations. Inverse paths observations are independent of direct path observations. The process described in section 4.2.3 to cast positive votes for different constraints is duplicated to be applied also with inverse paths. In general, this causes that the decorated instance dictionary grows more when `inverse_paths` is active, so the memory usage increases too.

The events triggered during the execution of the FT are propagated to the rest of remaining modules because, in general, more constraints need to be computed.

FIGURE 4.20: Python code - Disabling all-compliant mode

```

1
2  from shexer.shaper import Shaper
3
4  shaper = Shaper(all_classes_mode=True,
5                  graph_file_input="/path/local_file.nt",
6                  all_instances_are_compliant_mode=False)
7
8  print(shaper.shex_graph(string_output=True))
9

```

FIGURE 4.21: Python code - Keeping empty shapes

```

1
2  from shexer.shaper import Shaper
3
4  shaper = Shaper(all_classes_mode=True,
5                  graph_file_input="/path/local_file.nt",
6                  remove_empty_shapes=False)
7
8  print(shaper.shex_graph(string_output=True))
9

```

4.4.8 All-compliant mode

By default, sheXer generates shapes that comply with every instance used to build them. This is achieved by modifying cardinalities of conflictive constraints, so they include a zero-case (cf. section 4.2.4). However, those users who want to use the produced shapes as drafts may not be interested in getting shapes with perfect conformance among their instances. They may prefer instead to get constraints with more accurate cardinality information. To choose this option, one must set the parameter `all_instances_are_compliant_mode = False`, as shown in Figure 4.20

4.4.9 Removing empty shapes

By default, sheXer avoids producing results containing empty shapes, except when those shapes are part of a list of target classes or a shape map. This behavior can be changed with the parameter `remove_empty_shapes` as it is shown in Figure 4.21.

Empty shapes can be informative in several ways. On the one hand, the presence of an empty shape let the user know that a class (or the results of a certain node selector) does not have any instance, or those instances do not have features observed in a proportion of at least θ_U . On the other hand, the presence of an empty shape in a schema could be useful to produce more precise constraints.

Let us suppose that an execution of sheXer produce an empty shape $\langle Person \rangle$. Let us suppose that it also produces a shape $\langle Company \rangle$ with a constraint $c = (ex:worker @ \langle Person \rangle +)$. Even if the user does not know what a node $\langle Person \rangle$ should be like, the shape label itself raises a hint about the type of content expected to be found when a node $\langle Company \rangle$ uses the property `ex:worker`. If the $\langle Person \rangle$ shape is removed, then c should be modified, resulting in a constraint $c' = (ex:worker IRI +)$. The nodes conforming with $\langle Company \rangle$ from a pure validation point of view would be the same. However, from the documentation point of view, c' offers less information than c .

FIGURE 4.22: Python code - Disabling exact cardinality

```

1
2  from shexer.shaper import Shaper
3
4  shaper = Shaper(all_classes_mode=True,
5                  graph_file_input="/path/local_file.nt",
6                  disable_exact_cardinality=True)
7
8  print(shaper.shex_graph(string_output=True))
9

```

4.4.10 Cardinality prioritization

By default, sheXer aims to produce constraints with maximum agreement among the target entities but also as precise as possible. This process is described in Algorithm 2.

However, exact cardinalities can be hard guesses for certain execution context. Setting the parameter `disable_exact_cardinality` as shown in Figure 4.22, sheXer avoids the generation of constraints with an exact number of occurrences, except when this number is exactly 1. That is, the only possible cardinalities for any constraint when this mode is active are `{1}`, `+`, `?`, and `*`.

4.4.11 Adaptation to Wikidata model

4.4.11.1 Readable results

Wikidata is nowadays a key project for the LD community. As stated in previous sections, Wikidata uses its own ontology, which defines opaque URIs¹⁷. Classes, properties, and entities' URIs consist of a letter plus a numeric identifier which is unrelated with the concept represented by the URI.

For that reason, shapes based on Wikidata elements are usually hard to read for humans. When working with Wikidata content, sheXer can be told to generate comments in which each property or entity used in a constraint is mapped to a human readable label in a text comment. One must set the parameter `wikidata_annotation = True` for such a goal. Also, those comments will be generated using `rdfs:comment` annotations. This allows the user to translate the results into some other ShEx or SHACL syntax using third-party application without losing this information.

4.4.11.2 Handling qualifiers

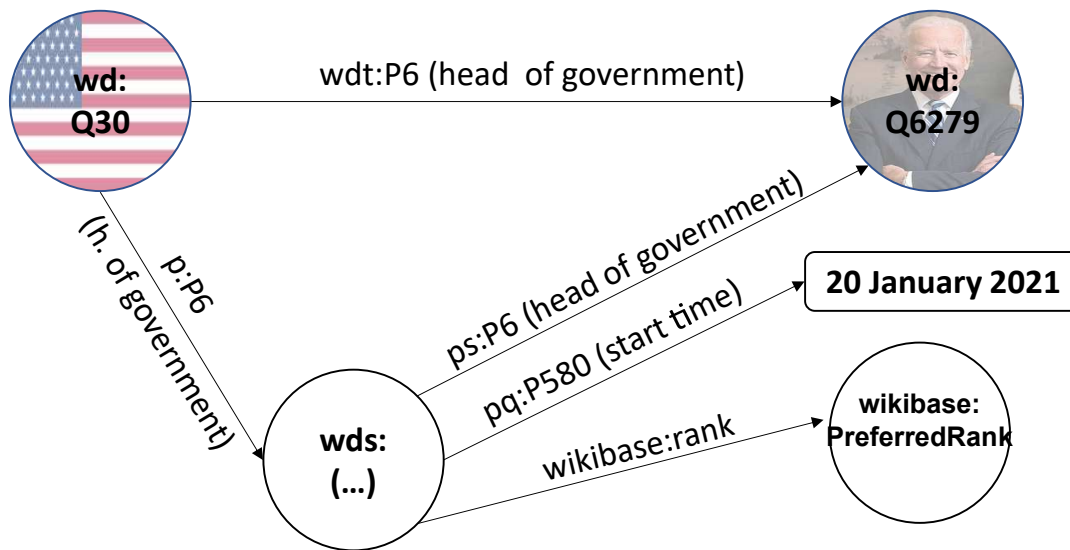
The content exposed in Wikidata's endpoint represents most of the relations in two different ways. The first one consists of the so-called *direct-properties*¹⁸. These properties are used as predicates in triples where the subject is a 'Q' element and the object is either a 'Q' element or a literal. The content offered using direct properties do not include all the content uploaded to Wikidata:

- References and qualifiers are not represented using direct properties. These are key elements in Wikidata's ecosystem. On the one hand, a reference about

¹⁷"Opaque URIs are resource identifiers which are not intended to represent terms in a natural language" [264]. They are useful in multilingual contexts such as Wikidata when it is preferred to not have language bias.

¹⁸Cf. <https://heardlibrary.github.io/digital-scholarship/lod/wikibase/> Accessed in 2022/05/03.

FIGURE 4.23: ('Q30 - United states', 'P6 - head of government', 'Q6279 - Joe Biden') using both direct and non-direct Wikidata properties



a statement allow to check that the referenced piece of information was obtained from a reputable source. On the other hand, qualifiers allow to nuance the statements. For instance, qualifiers could define a validity period for a certain relation. Representing both references and qualifiers in RDF requires using some kind of reification schema, which is not compatible with direct properties.

- Only the so-called preferred statements¹⁹ are represented with direct properties. Preferred statements in Wikidata help users to quickly understand the most up to date, current, correct, or, in short, *preferred* statement for a given subject and property. For instance, there are many statements linking 'Q30 - United states' with some elements using the property 'P6 - head of government'. However, among those relations, there is just one preferred statement, which is the one linked to the current US head of Government.

Wikidata's SPARQL endpoint also offer information using non-direct properties. Such properties do include qualifiers and references. For that, the actual subject and object of any notion are not directly linked in a triple. A *statement node* is placed between these two elements, so the subject points to the statement node with an indirect property, and the statement node points to the object with another indirect property. With this schema, each relation can be associated with qualifiers and references by linking such elements with the statement node. Since statement nodes have its own unique ID, they are not blank nodes. However, they are used as so, as they do not represent actual entities, but are used as a convenient structure to nuance pieces of knowledge.

As a running example, in Figure 4.23, we show the notion ('Q30 - United states', 'P6 - head of government', 'Q6279 - Joe Biden') represented both in direct and non-direct style.

When one wants to extract a Wikidata shape based on direct statements, the Wikidata properties used for reification can be noisy. sheXer offers a parameter `namespaces_to_ignore` to deal with this situation. This parameter expects a list of

¹⁹Cf. <https://www.wikidata.org/wiki/Help:Ranking> Accessed in 2022/05/03.

FIGURE 4.24: Python code - Extracting Wikidata shapes with direct properties

```

1
2  from shexer.shaper import Shaper
3
4  # The namespaces specified in this list ignore Wikidata's
5  # non-direct properties. Some other namespaces could be
6  # specified to exclude some other properties which are not
7  # Wikidata direct ones from appearing in the shapes, such
8  # as "http://www.w3.org/2004/02/skos/core#",
9  # "http://schema.org/", or "http://wikiba.se/ontology#"
10 namespaces_to_ignore = [
11     "http://www.wikidata.org/prop/",
12     "http://www.wikidata.org/prop/direct-normalized/"
13 ]
14 i_p="http://www.wikidata.org/prop/direct/P31"
15
16 # Q85795487 --> RNA vaccine
17 shaper = Shaper(target_classes=[
18     "http://www.wikidata.org/entity/Q85795487"
19 ],
20     url_endpoint="https://query.wikidata.org/sparql",
21     instantiation_property=i_p,
22     namespaces_to_ignore=namespaces_to_ignore)
23
24 print(shaper.shex_graph(string_output=True))
25

```

namespaces. sheXer will ignore triples whose predicate is a direct child of any of the namespaces indicated. With this, reification properties can be ignored, while direct ones can be parsed. A configuration example to extract Wikidata shapes excluding non-direct properties is shown in Figure 4.24.

In contrast, a user could be interested in the qualifiers. In such a case, sheXer can adapt to the Wikidata schema by using several parameters:

- `shape_qualifiers_mode`. This parameter must be set to `True`.
- `namespaces_for_qualifier_props`. This parameter expects a list of namespaces. This list must contain those namespaces used to link entities with node statements. With the example shown in Figure 4.23, that list should contain a single element, which would be the *p* namespace. This is the usual configuration to work with standard Wikidata content.
- `namespaces_to_ignore` (optional). This parameter could be set with the direct-properties namespace. In this case, the shapes produced would contain only non-direct properties.

When this configuration is active, sheXer understands that every node found as object in a triple whose predicate is a property of one of the namespaces defined in `namespaces_for_qualifier_props` is a statement node. Regardless of its declared type (if any), sheXer will create a shape for each group of statement nodes pointed with a same property. With this, for a certain non-direct property, the user can know which are the usual qualifiers and references used to describe its statement node. An example of such a configuration is shown in Figure 4.25.

FIGURE 4.25: Python code - Extracting shapes for Wikidata qualifiers

```
1
2 from shexer.shaper import Shaper
3
4 namespaces_to_ignore = [
5     "http://www.wikidata.org/prop/direct",
6     "http://www.wikidata.org/prop/direct-normalized/"
7 ]
8 i_p="http://www.wikidata.org/prop/direct/P31"
9
10
11 # This example consumes Wikidata's endpoint. In order to
12 # retrieve triples for the statement nodes, the depth to
13 # explore the subgraph should be at least 2. Otherwise,
14 # the shapes created for statement nodes will be created
15 # but will be empty.
16 shaper = Shaper(target_classes=[
17     "http://www.wikidata.org/entity/Q85795487"
18 ],
19     url_endpoint="https://query.wikidata.org/sparql",
20     instantiation_property=i_p,
21     namespaces_to_ignore=namespaces_to_ignore,
22     wikidata_annotation=True,
23     depth_for_building_subgraph=2,
24     shape_qualifiers_mode=True,
25     namespaces_for_qualifier_props=[
26     "http://www.wikidata.org/prop/"
27 ])
28
29 print(shaper.shex_graph(string_output=True))
30
```


4.5 Related work

Several approaches to automatically extract shapes have been proposed. The instance-based proposals, such as sheXer, extract shapes from KG mining. In contrast, ontology-based approaches compute ontologies T-BOX knowledge to produce shapes.

The closest work to sheXer is Shape Designer [256]. Shape Designer consists of a tool to perform automatic extraction of shapes with KG mining. Both sheXer and Shape Designer support ShEx and SHACL, and both keep an internal score of how trustworthy a given constraint can be w.r.t. how frequently is it supported by the nodes used to extract it. However, there are fundamental differences between these two approaches. Shape Designer is integrated with a graphic tool and aims to produce shapes that are not intended to be definitive. The tool extracts candidate shapes that the user can customize later. In opposition, sheXer aims to produce shapes as accurately as possible and does not necessarily include human intervention in its workflow. Also, sheXer and Shape Designer offer different approaches to solve constraints including IRIs. Shape Designer uses either the macro `IRI` or value sets that can restrict the possible IRIs to a string pattern. sheXer can produce actual shape inter-linkage, i.e., triple constraints whose object is another shape label.

The system proposed in [257] uses a machine learning approach to generate SHACL shapes associated with classes. The authors choose class-property combinations and associate them to two types of constraints: cardinality and range. Cardinality refers to the minimum and maximum occurrences. Range refers to the type of object, which is one of `sh:IRI`, `sh:BlankNode`, `sh:BlankNodeOrIRI`, `sh:Literal`, or specific literal types. Both types of constraints have a finite set of possible final values, so the approach is formulated in terms of a classification problem. Once all pairs have been associated with their constraints, the constraints of a given class c are all merged to produce a SHACL shape associated to c .

The approach presented in [112] transforms abstract semantic profiles generated by ABSTAT [225, 265] into SHACL shapes. ABSTAT generates abstract semantic profiles of KGs, which essentially consist of statistics regarding relations between types and data-types. These statistics are then used to generate SHACL shapes. Since these profiles do not handle the concept of shape, the tool does not perform shape inter-linkage. However, with such an approach, shape-interlinkage could be produced in case there is a one-to-one relation between each class and each shape, so exact mappings can be created. This proposal is the only approach (apart from sheXer) that we are aware of it is able to produce inverse paths. This system also computes the frequency in which a certain feature is observed among a group of nodes. This information is used to exclude part of the information generated by ABSTAT. As already stated, sheXer uses it for the same purpose, but also to sort the constraints w.r.t. its trustworthiness score, and to produce text comments that could be relevant for the user. The constraints regarding cardinality are expressed in terms of minimum and maximum occurrences outputted by ABSTAT.

Regarding automatic conversions between ontologies and shapes, the authors in [258] propose using Ontology Design Patterns (ODP) to obtain SHACL shapes. However, no actual mappings between SHACL and ODP are proposed. In [266], the authors provide an extended argument for using ODPs to produce SHACL shapes. ODPs, unlike ontologies, are modular and naturally bounded to application contexts, can easily evolve with use cases, and can combine concepts and relationships of multiple ontologies. ODPs are adequate tools to describe expected schemata in specific RDF graphs. Therefore, mappings between ODPs to SHACL shapes allows for reusing ODPs for validation tasks. In [267], SHACL and OWL are thoroughly

compared in terms of meaning and expressiveness. The authors also provide mappings between OWL and SHACL. These mappings can be used by proposals that aim to extract shapes from T-BOX content.

Astrea [259] is a tool to perform automatic extraction of shapes from ontologies. The authors produce SHACL content by mapping ontology patterns into SHACL constructions. Astrea is a publicly available tool based on the mappings of Astrea-KG²⁰. Astrea-KG's content allows generating SHACL shapes with an expressiveness that includes 60% of the total constraints available in SHACL.

SHACLerarer, a method to learn SHACL constraints based on Inverse Open Path rules (IOP), is presented in [260]. SHACLerarer adapts Open Path Rule Learner (OPRL) [268] to extract IOP rules, which can be translated to SHACL. SHACLerarer works with rules between entities in a KG, which causes that the shapes obtained do not contain constraints related to literals. Same as sheXer, the authors of SHACLerarer evaluate its system using large RDF sources. SHACLerarer proves to be able to handle a slice of DBpedia containing 11,498M triples in a 66GB RAM environment.

SHACLGEN²¹ is a public Python library to extract SHACL shapes. It can handle both instance data and ontologies. However, it uses an approach that requires to load the input data in a triplestore, which leads to scalability issues when handling large input graphs. The private tool TopBraid Composer²² also supports the extraction of SHACL from ontologies and instance data.

In [269], a method to generate SHACL and ShEx shapes from R2RML documents is presented. R2RML is a W3C standard language to enable automatic translation from Relational databases to RDF documents. R2RML files are written also in RDF, and they describe the mappings from the database's model to the RDF data model. Since the generation of shapes is based on R2RML, this approach can be applied only in KGs whose genesis is a mapped relational database. However, it achieves excellent conformance with the target data model.

A similar work is presented in [270]. In this case, authors are able to generate SHACL shapes by using mappings from RDF Mapping Language (RML) [271]. RML is an extension of R2RML. However, unlike R2RML, RML not only allow to transform relational database schemata into RDF, but it can handle different input sources. With this, the approach presented in [270] has a broader scope than the proposal in [269].

Some other previous works extract different schema notions from RDF graphs. In [272], the authors present an approach to extract frequent graph patterns conceptually similar to shapes, whose aim is to characterize the content of RDF triple-stores. The patterns are represented using an adaptation of Deep-First Search code. The authors in [273] extract Knowledge Patterns from KGs. These patterns are expressed in OWL and characterize classes by detecting their frequent properties and providing an adequate range for them.

4.6 Conclusions

At the beginning of this chapter, we stated the following Research Question:

- **RQ2:** How can we produce shapes by mining RDF triples?

²⁰Endpoint to query Astrea KG: <https://astrea.helio.linkeddata.es/sparql> Accessed in 2022/05/03.

²¹<https://pypi.org/project/shaclgen/> Accessed in 2022/05/03.

²²<https://www.topquadrant.com/from-owl-to-shacl-in-an-automated-way/> Accessed in 2022/05/03.

To provide an answer to this question, we have developed sheXer, a system to perform automatic shape extraction based on generalization of instance knowledge. Our proposal extracts shapes by mining the neighborhood of some seed target nodes. Each extracted constraint is qualified with a trustworthiness score that allows for filtering infrequent elements, sorting results, and providing extra information using textual comments.

Despite there are other existing approaches to perform automatic shape extraction, sheXer fits better in our thesis context because its unique combination of features. The features of sheXer that makes it a suitable system to be use in our context are:

- It is instance-based, which is a prerequisite in our scenario.
- It allows for processing large real-world datasets.
- It performs shape inter-linkage, which allows for producing more specific constraints.
- It can extract inverse paths. This let us capture more knowledge about the neighborhood of the target nodes of a shape.
- Its trustworthiness score allows us to discard features rarely occurring. It also allows us place the most trustworthy constraints at the top of a shape. Both goals are key to our scenario, as shapes obtained by instance-based extractors could be too long and noisy in heterogeneous graphs.
- The possibility of producing ShEx and SHACL content allow us to build a system whose output is useful for the users of both languages.

Our system is based on an iterative mining strategy that avoids loading in main memory the entire KG whenever this is possible. The execution time and the peak of memory usage of our proposal have a linear relation with the number of instances relevant for the extraction process. However, we have shown that the shapes obtained using large amounts of instances tend to converge with shapes obtained using a low but representative number of elements. An adequate instance limit has a significant positive effect on memory usage and execution time, and little effect on the results' quality.

The experiments also reveal linear relations between several parameters of the input and execution times, including the number of triples relevant for the process, the dataset size, or the number of target shapes. Those relations could become scalability issues when dealing with too large data sources. However, as discussed in section 4.3.4.1, it is feasible to produce an implementation of sheXer using parallel computing approaches.

Beyond the context of our thesis, sheXer has proven to be a competitive approach for the general task of automatic shape extraction. As shown in section 4.4, our current implementation of sheXer can be adapted to many scenarios and user preferences.

We have publicly released a public and free to use Python library which implements the system described in this chapter.

4.6.1 Future work

Our current implementation of sheXer already fulfills the requirements for the general goals of the thesis. However, several future research lines emerge from our work:

- Producing and evaluating an implementation of sheXer using parallel computing based on the proposals discussed in section 4.3.4.1.
- Including ontological (T-BOX) information in sheXer’s workflow. The system could be improved in several ways with such a feature:
 - The shapes created could include constraints with an instance-based or an ontological-based genesis.
 - Instance-based constraints could be evaluated before producing the final results. Those causing some ontological violation could be adapted or discarded.
 - It could be feasible to generate hierarchical results ontology-based. For instance, our current implementation of sheXer could generate a shape *<PersonShape>* for instances of *:Person*, and a *<ITWorkerShape>* shape for instances of *:ITWorker*. These two shapes could potentially share many properties. It could be feasible to write *<ITWorkerShape>* as an extension of *<PersonShape>* in case it can be ontologically determined that the class *:ITWorker* is a specialization of the class *:Person*.
- Working on a public benchmark to evaluate proposals for automatic shape extraction, both on performance terms (execution time, memory usage) and quality of results. The existence of such a benchmark is key to fairly compare existing approaches.
- sheXer has been designed to work with RDF data. However, the need of describing and validating schemata is also frequent in other types of KGs, such as the property graphs described in section 2.3. Thus, we also plan to adapt sheXer’s core ideas to other graph models.

Chapter 5

Mining shapes from social media

5.1 Introduction

In this chapter, we aim to provide an answer for the following Research Question:

- **RQ1:** How can we automatically extract shapes from social media content?

To the best of our knowledge, there is no system able to extract shapes from social media content. The closest existing proposal is described in [22]. This approach can extract SHACL content from textual descriptions of shape constraints, such as “Every user has exactly one username” or “Each student has at least one subject enrolled”. However, such type of content could be hard to find in social media. There are, however, systems which can perform some tasks that, connected in a pipeline, can generate shapes by parsing social media content. Specifically, there are two tasks which could be combined to extract shapes from social content: 1) the automatic extraction of triples from natural language, and 2) the automatic extraction of shapes from RDF content.

Extracting triples from natural language requires integrating the output from several subtasks. Some of those subtasks are NER, EL, and RE.

NER consists in detecting mentions of entities in plain text. In EL stages, the entities detected with NER are associated to unique identifiers (URIs). Finally, RE is performed to discover relations between those entities, which, in turn, should also be associated to URIs (properties). Some existing systems are specialized in solving NER [274, 275], EL [125, 276, 277], or RE [278, 279] problems, while some others provide complete solutions for the entire process [280–284].

The problem of extracting shapes from RDF content has been thoroughly studied in chapter 4. To that extent, we reviewed the existing approaches and developed and evaluated the library sheXer, which is an instance-based shape extractor.

Therefore, we propose a novel architecture that integrates subsystems of triple extraction from natural language and shape extraction from RDF content. We have named it *Shape Extraction from Instance Text Mining Architecture* (SEITMA). Implementations of this architecture can be used to perform the tasks associated to **RQ3**. SEITMA describes how to integrate the mentioned subsystems to produce using examples of instances described in natural language. Thus, SEITMA implementations are suitable candidates to extract shapes from textual content found in social media.

As a first use case for SEITMA, we chose the extraction of shapes from Wikipedia abstracts of the English Wikipedia chapter. We have implemented two prototypes following SEITMA specifications that were able to perform such a task. Both use sheXer to produce the final shapes, but they implement different approaches to extract triples from the Wikipedia abstracts. The shapes extracted are associated to important classes in DBpedia, which are determined using the ClassRank algorithm.

We also use important instances of those classes, which are determined using PageRank. The combination of ClassRank and PageRank allows us to locate a group of reasonably central entities with presumably more than enough text to be used to extract shapes for their classes. The input provided to the SEITMA prototypes consists of a set of Wikipedia abstracts associated to those target DBpedia entities.

Our experiment with Wikipedia abstracts is a proof of concept to 1) prove the feasibility of SEITMA proposals, and 2) discuss the strengths, weaknesses, and potential uses of different SEITMA implementations. We selected Wikipedia as the target source of this experiment for several reasons:

- Wikipedia is a public well-known corpus of natural language. Despite being an example of social media, its community-moderated encyclopedic-like writing style causes it to have higher quality standards than many other user-generated text corpora. This quality is a desirable feature to successfully extract triples from the text.
- Each Wikipedia page contains knowledge associated to a main entity, which is the title of the Wikipedia page. More specifically, each abstract in a Wikipedia page is supposed to contain just the core elements describing such entity. This content matches perfectly the SEITMA expected input, as the architecture must be fed with descriptions of different examples of a certain class. The concise nature of the Wikipedia abstracts makes them an excellent corpus of short entity descriptions. Those descriptions are expected to contain features (relations) that are likely to be found in abstracts of similar entities (instances of the same class).
- Most Wikipedia pages can be trivially associated with a single DBpedia URI that stands for the same entity described in the Wikipedia page. The relation between these two projects allows us to easily locate text descriptions of entities associated to important classes in DBpedia.

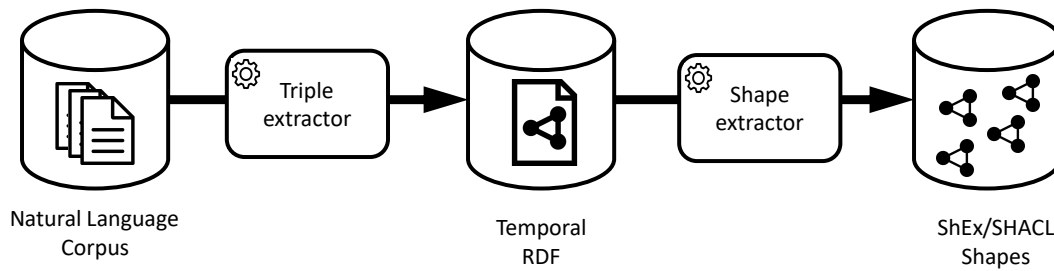
In this chapter, we describe in a detailed way our two prototypes and discuss the results of the experiments conducted with them.

The potential of the shapes that could be obtained with SEITMA implementations goes beyond the pure description or validation purposes usually associated to RDF shapes. The shapes obtained, as a formally structured expression of conceptual knowledge, could be used for a variety of practical purposes which can be beneficial in contexts which are not primarily related to RDF environments nor require RDF-skilled users. Examples of systems that could be built on top of the shapes generated by a SEITMA implementation are template generators, automatic text classifiers, or editing tools with content suggestion. In this chapter, we also introduce and discuss several use cases that were not explored during our experiments. Some of the use cases require shapes extracted from Wikipedia, but some others are framed in scenarios involving different text sources.

This chapter's content is organized as follows:

- In section 5.2, we describe the SEITMA core proposals.
- In section 5.3, we describe our two SEITMA prototypes.
- We start section 5.4 describing our experimental setup. Then, we present and discuss the obtained results. We also compare our prototypes to each other and discuss their potential.

FIGURE 5.1: SEITMA core



- In section 5.5, we introduce different use cases which can benefit from shapes produced using a SEITMA implementation.
- In section 5.6, we provide a review of related work. We focus on those approaches which perform tasks similar to SEITMA and make emphasis on systems that extract triples from natural language.
- Finally, in section 5.7, we expose the conclusions of this chapter and describe lines of future work.

5.2 Architecture description

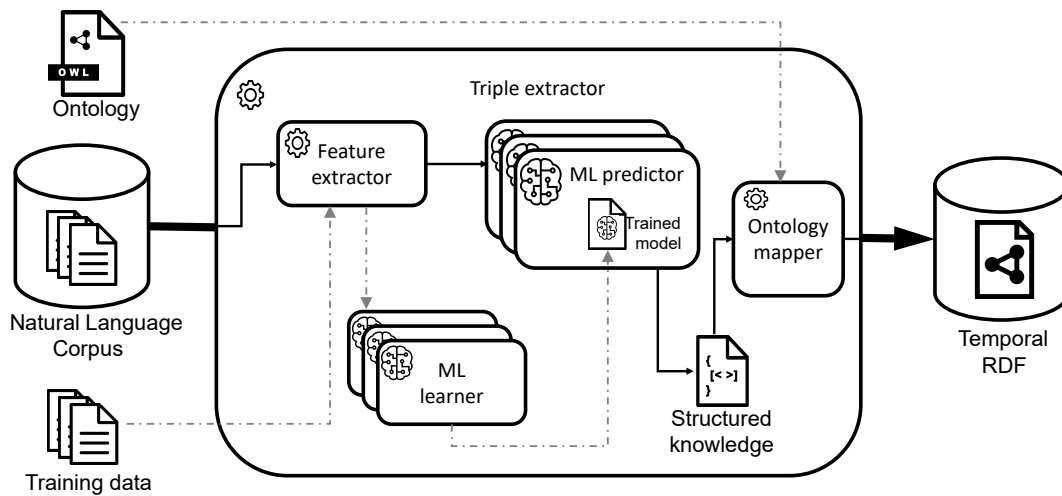
SEITMA is an architecture for the extraction of shapes associated to classes through the generalization of knowledge found at instance level. A SEITMA implementation must be fed with different pieces of text containing information about some target entities. Those target entities must be associated with a URI and should have, at least, a type (an RDF class). SEITMA should transform the input text into RDF triples. The generated RDF content should contain triples in which the URIs associated to the target entities are used (either as objects or subjects). Then, SEITMA would produce shapes associated with the classes of those entities.

In this workflow, the instance-level knowledge, which is transformed from raw text to RDF, is used as an example of a class occurrence. The features associated with a class are determined by generalizing those examples, i.e., each shape produced with SEITMA describes the features observed among a class's instances.

The SEITMA core is graphically represented in Figure 5.1. SEITMA proposes to combine two potentially independent software elements to perform shape extraction from pieces of natural language:

- **C1, Triple extractor.** A component to extract RDF from a natural language corpus. Two kinds of triples should be extracted:
 - **Typings.** The entities that are going to be used as targets (or seeds) to extract a certain shape must have the same type.
 - **A-BOX knowledge.** The suitability of the shapes produced is aligned to the amount and quality of the triples produced at instance level. More specifically, a SEITMA implementation should generate as many truthful triples using typed entities as possible.
- **C2, Shape extractor.** This component will consume the RDF content generated in the prior stage, group entities by class, and produce shapes associated with those classes.

FIGURE 5.2: SEITMA's C1. General solution for an ML approach



Both components can be implemented using a wide variety of approaches, so both may involve a complex internal architecture. For example, *C1* could be based in a usual combination of NER, EL, and RE analyses to extract entities and relations in the target text. Then, a mapper could be used to convert the output of those processes into triples using URIs from some target ontologies.

Figure 5.2 shows an example specification to implement SEITMA's *C1* based on ML approaches. In general, such a system would require three different input elements: 1) Training data for prediction models, 2) target text to extract knowledge, and 3) ontologies so the knowledge extracted can be expressed as triples using controlled vocabularies. The workflow of this example system consists of the following steps:

1. The training data is sent to the *Feature Extractor*.
2. The Feature Extractor transforms the positive and negative examples of the training data into the sets of feature representations expected by ML algorithms.
3. The adapted training data is sent to some *ML learners*. These learners produce trained models for identifying entities/relations in new data. Note that this stage can involve one to many *ML learners* and can produce *one to many* trained models, depending on the algorithms used.
4. The target texts are sent to the Feature Extractor, so they can be also transformed into sets of input features.
5. The sets of features representing the target texts are sent to the *ML predictors*. These components are supposed to use the models produced by the ML learners to perform actual predictions for the input features.
6. The ML predictors should produce results in a structured form. This content, however, may not consist of actual triples. The ML predictors output is sent to an *Ontology mapper* to be transformed into actual RDF which uses URIs from target ontologies.
7. The Ontology mapper should also produce typing triples for the entity URIs generated.

Note that Figure 5.2 is just a general solution involving ML. For example, it may not be needed to perform the mapping stage after the prediction stage. Instead, the mapping could be partially or totally performed during the feature extraction stage in case the ML predictors were trained to produce results using URIs from the target ontologies. .

The type of RDF content produced by *C1* should contain instance-level knowledge. For this reason, in every case, the *C2* component should be implemented using an instance-based shape extractor. As explained in chapter 4, the existing automatic shape extractors could be divided in two broad categories: instance-based and ontology-based. The later ones analyze T-BOX knowledge to produce shapes. These shapes are built using direct mappings between ontology restrictions and elements in the resulting shapes. In contrast, instance-based approaches aim to produce shapes based on generalization of examples. They explore the A-BOX neighborhood of some target nodes, so they can extract constraints based on common features observed on those neighborhoods.

5.3 Prototypes

We have implemented two prototypes of SEITMA, named SEITMA-L and SEITMA-F. These two prototypes aim at the same goal: being able to extract shapes associated to DBpedia classes using Wikipedia abstracts as input text.

Both systems use sheXer to transform RDF triples into actual shapes, but they follow different approaches to implement the SEITMA's triple extractor submodule. In this section, we describe the internal details of SEITMA-L and SEITMA-F. We make especial emphasis in the triple generation stage, as the internal complexity of the shape generation stage was fully described in Chapter 4.

5.3.1 Prototype 1: SEITMA-L

SEITMA-L is based on the ML architecture described in Figure 5.2. The shape extractor role is played by sheXer. The triple extractor role is played by an ad-hoc implementation of a system based on the proposals of the work *Language-Agnostic Relation Extraction from Wikipedia Abstracts* (LAREWA)¹ [285]. SEITMA-L has been implemented using Python and is publicly available in a GitHub repository².

In this section, we first enumerate and explain the key proposals of LAREWA and then describe SEITMA-L.

5.3.1.1 LAREWA

LAREWA consists in an approach to perform NEL and RE in Wikipedia abstracts. The outputs of this system can be used to automatically generate RDF triples from Wikipedia's content. LAREWA's proposals are based on an ML approach trained with examples of equivalences between entities mentioned in Wikipedia abstracts and triples in DBpedia. The examples are used to predict relations between an entity mentioned in an abstract and the entity mentioned in the abstract's title. LAREWA's main premise is that similar contexts may express similar relations.

Authors in [285] indicate that they were able to generate 1.6M relations (triples) that were not originally in DBpedia with a precision of 95% using their system.

¹The acronym *LAREWA* has not been coined by the authors of [285]. We introduce this acronym to improve the readability of this chapter, as this work will be mentioned several times.

²https://github.com/DaniFdezAlvarez/wikipedia_shexer Accessed in 2022/05/03.

LAREWA's model to represent a text context is based on numerical or boolean variables that are language-agnostic. Thus, potentially, this approach could be used for any Wikipedia chapter with a corresponding DBpedia chapter in the same language. LAREWA's publication indicates how to obtain data to train the ML models and how to generate new triples with these models.

Finding training data - Authors in [285] do not use human-labeled data to train LAREWA's models. Instead, they use distant supervision, i.e., automatic alignments between text pieces and a database of facts [200]. With such an approach, they can find pieces of knowledge represented in text form in Wikipedia and in RDF form in DBpedia. They use these equivalences as positive examples to train the ML algorithms. For a given Wikipedia abstract of an entity e , this process involves the following steps:

1. They perform NER to locate entities mentioned in the abstract. This process is trivial, as it merely consists in finding *wikilinks*³ w_i . The text anchored in the wikilink is the label of the named entity.
2. The URL of e and the links of each w_i page are mapped to their corresponding DBpedia URI. This EL process can be trivially performed too, as the last part of the path of a Wikipedia URL is identical to the last part of its corresponding DBpedia URI, and all DBpedia entities belong to the same namespace.
3. For each combination (e, w_i) , it is checked whether there exists a triple in DBpedia such as (e, r, w_i) . If it does exist, then the text context surrounding the mention of w_i in the Wikipedia abstract of e is considered a **positive** example to express the relation r .
4. Negative examples of a relation r are found using ontological knowledge and *local closed world assumption* [286]. It is assumed that a certain relation is complete in its local context. Hence, for a relation r and a certain context C , it is assumed that no other entities are linked with r unless this relation is stated in C . Gathering negative examples involves the following steps:
 - (a) For a relation r with at least one positive example (e, r, w_a) , it is checked whether there is any other entity w_i mentioned in e 's abstract compatible with r according to the *rdf:range* defined for r in the DBpedia ontology.
 - (b) For each w_i that meets the previous condition, if it is true that there is not a triple in DBpedia such as (e, r, w_i) , then the text context surrounding the mention of w_i in the Wikipedia abstract of e is considered a **negative** example to express the relation r .

The examples (positive and negative) found for a relation r in a certain abstract are called *candidates of r* .

The process of gathering positive and negative examples within a Wikipedia abstract can be easily explained with an example. The United States' Wikipedia page (entity *dbr:United_States* in DBpedia) contains, among others, mentions to Washington D.C. (*dbr:Washington,_D.C.*), New York (*dbr:New_York_City*), and the Moon (*dbr:Moon*). In DBpedia, the triple $(dbr:United_States, dbo:capital,$

³*Wikilinks* are Wikipedia hyper-links that lead to another Wikipedia page.

dbr:Washington,_D.C.) exists. Then, we assume that the context surrounding the mention of Washington D.C. in United States' abstract is a positive example of the relation *dbo:capital*.

To find negative examples for this relation, we look for any mention w_i whose type is compatible with the ontological description of *dbo:capital*. In this case, DBpedia ontology states that the range of *dbo:capital* is *dbo:City*. Therefore, we need to find mentions whose DBpedia URI is declared as an instance of *dbo:City* or any of its subclasses. For each mention meeting this condition, if there is not a DBpedia triple such as $(dbr:United_States, dbo:capital, w_i)$, then w_i can be used as negative example. *dbr:New_York_City* is instance of *dbo:City*, therefore *dbr:New_York_City*'s textual context is used as a negative example of the relation *dbo:capital*. In contrast, *dbr:Moon*, which is instance of *dbo:Planet*, *dbo:Location*, and *dbo:Place*, is not used as a negative example. This mention is excluded from any consideration w.r.t. to the *dbo:capital* relation.

In [286], this procedure is applied over DBpedia properties having explicit domain and range definitions in DBpedia ontology.

Features - As already stated, LAREWA is based on a language agnostic ML approach. Each positive and negative example is mapped to a set of features that describe the context of a mention. The concept of *candidate* already described is used in several features. The features for describing a candidate are the following ones:

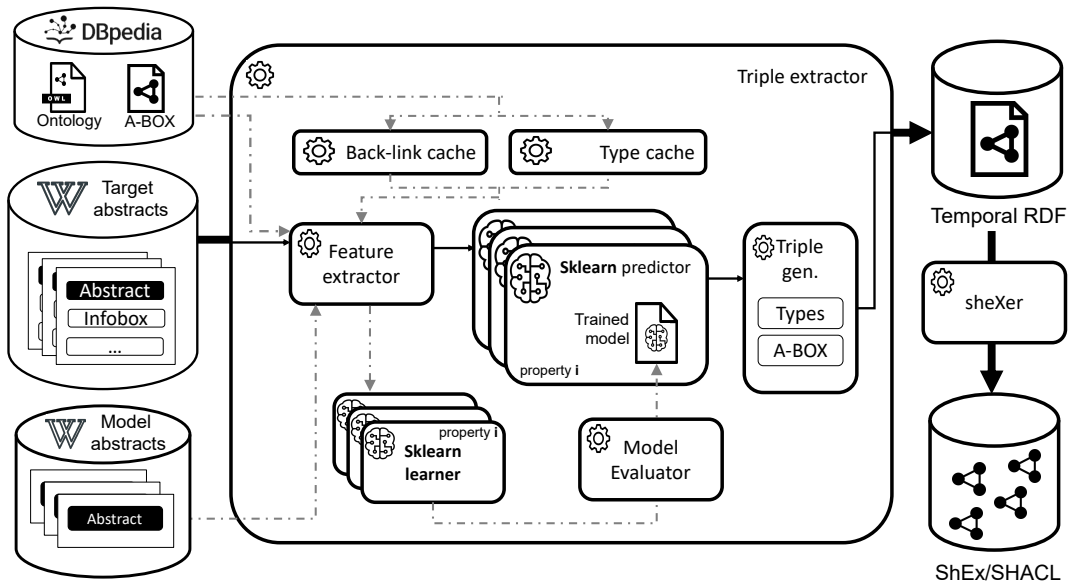
- F1. Total number of candidates in the abstract.
- F2. Total number of candidates in the sentence.
- F3. Relative position in the abstract w.r.t. other candidates.
- F4. Relative position in the sentence w.r.t. other candidates.
- F5. Total mentions in the sentence.
- F6. Relative position in the abstract w.r.t. other mentions.
- F7. Relative position in the sentence w.r.t. other mentions.
- F8. Relative position of the sentence w.r.t. other sentences.
- F9. Existence of a back-link in the mention's abstract to the target entity.

Each relevant mention in an abstract is represented as a vector containing these 9 features plus a boolean value that indicates whether it is a positive or a negative example.

Performing classifications - Once the training data has been obtained, it is used to train as many classifiers as properties are under analysis. Each classifier is binary and works with a single property r . These classifiers receive input vectors with the features enumerated in the previous subsection. They evaluate whether each vector is a positive example of the property r . Vectors found positive are used to produce an actual triple linking the abstract's title and the mentioned entity by means of r .

In [286] the authors just use those classifiers achieving a precision of at least 95% evaluated on the training data. From an original set of 395 candidate properties, they were able to train classifiers meeting this condition for 99 cases.

FIGURE 5.3: SEITMA-L: implementation based on a language-agnostic ML approach



With those 99 classifiers, they were able to generate 998,993 new relations, i.e., triples that were not originally in DBpedia.

The authors evaluated several ML algorithms, including Naive Bayes, RIPPER [287], Random Forests [288], Neural Networks, and Support Vector Machines (SVM) [139], and found that Random Forests were the best performers with their training data.

5.3.1.2 SEITMA-L implementation

The internal architecture of SEITMA-L's triple extractor is detailed in Figure 5.3.

Similarly to LAREWA, SEITMA-L's triple extractor works with a workflow consisting of two stages. In the first one, it collects training data and trains a number of ML classifiers. In the second one, it generates triples using those classifiers.

Stage 1: Training classifiers A number of target Wikipedia abstracts are selected to be used as model. These abstracts are sent to the *Feature extractor* module, which detects positive and negative examples using a system based on LAREWA. Our implementation differs slightly from LAREWA though:

- We compute relations with properties having an explicit domain *or* range declaration in DBpedia ontology. LAREWA is described to work just with those properties that have both domain *and* range explicit declarations.
- SEITMA-L generates two different classifiers for each relation r considered. LAREWA's classifier for a relation r work with triples such as (e, r, w_i) , where e is the entity whose abstract is being parsed and w_i is a mention within this abstract. SEITMA-L also trains a classifier to recognize triples such as (w_i, r, e) , i.e., triples where e is used as object instead of subject. This kind of triples are useful to obtain shapes with inverse paths.
- Our system uses two special caches to featurize a certain abstract in affordable time:

- **Back-link cache.** It is used to determine in $O(1)$ complexity which are the Wikipedia pages linked from a certain page. This cache is populated processing DBpedia *wikilinks*, i.e., DBpedia triples that indicate directed links between Wikipedia pages.
- **Type cache.** It is used to determine in $O(1)$ which are the declared types within the DBpedia ontology for a given entity. This cache is populated processing DBpedia typing triples.

For n relations considered, $2 \cdot n$ classifiers are trained. Those classifiers are sent to a *Model evaluator* module. This module is used to filter unreliable classifiers. A certain classifier must meet two conditions to be accepted:

- The data used to train it must contain a minimum number of positive and negative examples θ_n .
- Evaluated with a random split of the training data, the classifiers must reach at least a certain precision θ_p .

Several ML algorithms were evaluated to be used as classifiers. Using a representative sample of entities linked to important classes in Wikipedia⁴, and setting an arbitrary $\theta_p = 0.75$ we found that the algorithm producing better results was SVM. SVM outperformed the rest of algorithms both in total number of classifiers with at least θ_p precision, and average precision among the accepted classifiers.

Stage 2: Triple generation - Two types of inputs are required to perform the stage of triple generation. On the one hand, the Wikipedia abstracts of the set of entities E from which the triples are going to be extracted. On the other hand, a local file containing a partial DBpedia dump. This dump should include, at least, wikilinks and typing triples. This information is used to build the Back-link and Type-link caches, which are also used in this stage.

In this stage, target abstracts are sent to the Feature extractor to transform the candidate mentions into sets of features. This process has some key differences with the feature extraction performed in stage 1. In that stage, negative examples are generated w.r.t. to positive ones. For a certain abstract A_e of an entity e , and a certain relation r , there must be a positive example of r in A_e to look for negative examples of r in A_e .

In contrast, in stage 2, it is unknown whether a certain mention w_i is a positive example. The classifiers determine this *a posteriori*. Then, every mention compatible with a certain relation r in domain and range is used as candidate. An immediate consequence of this approach is that the number of candidates tend to be much higher. The less constrained is the domain and range definition of a certain property, the more noticeable is the increase of the number of candidates.

SEITMA-L generates candidates just for those relations that are associated to a valid classifier, i.e., a classifier meeting the minimal quality conditions specified with θ_n and θ_p .

Once the candidates are featurized, they are evaluated with their respective classifiers. The candidates classified as positive examples of a relation are used to generate RDF triples. This task is performed in the *Triple generator* module.

The Triple generator also outputs typing triples for the entities in the content generated using classes declared in the DBpedia ontology. Those typing triples allow the shape extractor to select which entities should be used to extract which shapes.

⁴The nature of this sample is described in section 5.4.

Shapes from triples - sheXer is used to extract shapes from the triples generated. sheXer is executed using `inverse_paths = True` and `all_classes_mode = True` (cf. section 4.4 for a detailed explanation of these working modes).

This configuration has two consequences. On the one hand, the shapes obtained can contain inverse paths/triple constraints, i.e., they can describe topological features where the focus node is the object of a triple. On the other hand, the system will output shapes for each class with at least an instance. Therefore, SEITMA-L will produce shapes related to the classes of the target abstracts, but also shapes for the classes of entities mentioned in those abstracts.

About input files and system requirements - SEITMA-L does not use any Application Programming Interface (API) nor remote endpoint. Every input is provided via local files. The required files are the following ones:

- **Wikipedia dump file** to get both the model and target abstracts to mine.
- **DBpedia partial dumps**, including:
 - **Typing triples** to build the Type cache.
 - **Wikilinks** to build the Back-link cache.
 - **Object-to-object relations**, to identify positive and negative candidates in stage 1.
 - **Class hierarchy** and **domain/range declarations** for elements within the DBpedia ontology, so negative examples in stage 1 and candidates in stage 2 can be identified.

Avoiding the use of any kind of on-line service has several effects.

On the positive side, it enables to perform big computations. Wikimedia public APIs apply temporal Internet Protocol (IP) bans after receiving too many requests from a certain IP address in a short time period. One cannot make extensive use of these services without a special permission from Wikimedia.

On the negative side, the hardware requirements to execute SEITMA-L increase, as one need to have enough disk space to store the dump files and enough Random Access Memory (RAM) memory to allocate some index structures. The two software pieces that demand more RAM are the Back-link cache and the Type cache.

The Wikipedia dump and the object-to-object link graph do not need to be allocated in main memory. These two structures are parsed iteratively by the Feature Extractor. That submodule keeps in main memory only the relevant information related to the target entities and their mentions.

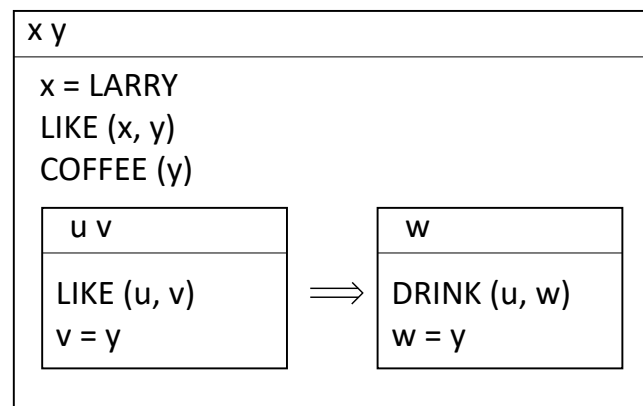
5.3.2 Prototype 2: SEITMA-F

As with SEITMA-L, SEITMA-F uses sheXer in the role of shape extractor. For the role of triple extractor, SEITMA-F uses an ad-hoc implementation based on the FRED system [176]. In this section, we will first describe FRED. Then, we will describe our prototype's architecture and workflow.

5.3.2.1 FRED

FRED is presented as a machine reader, i.e., a tool able to transform text in natural language to some sort of knowledge with a formal structure. Specifically, FRED transforms natural language into RDF/OWL ontologies.

FIGURE 5.4: DRS representation of “Larry likes coffee. If someone likes it, he drinks it.”



The graphs produced by FRED are designed according to Frame Semantics [289]. In this theory, a frame describes a system in which a certain element cannot be completely understood without the others. For instance, the complete meaning of a verb such as *eat* in a certain context requires to know circumstantial elements such as *what is being eaten* or *who/what is eating it*. FRED represents frames using OWL *n*-ary relations⁵ [290]. The frames’ roots are instances of some kind of event or situation.

FRED’s workflow consists of the following steps:

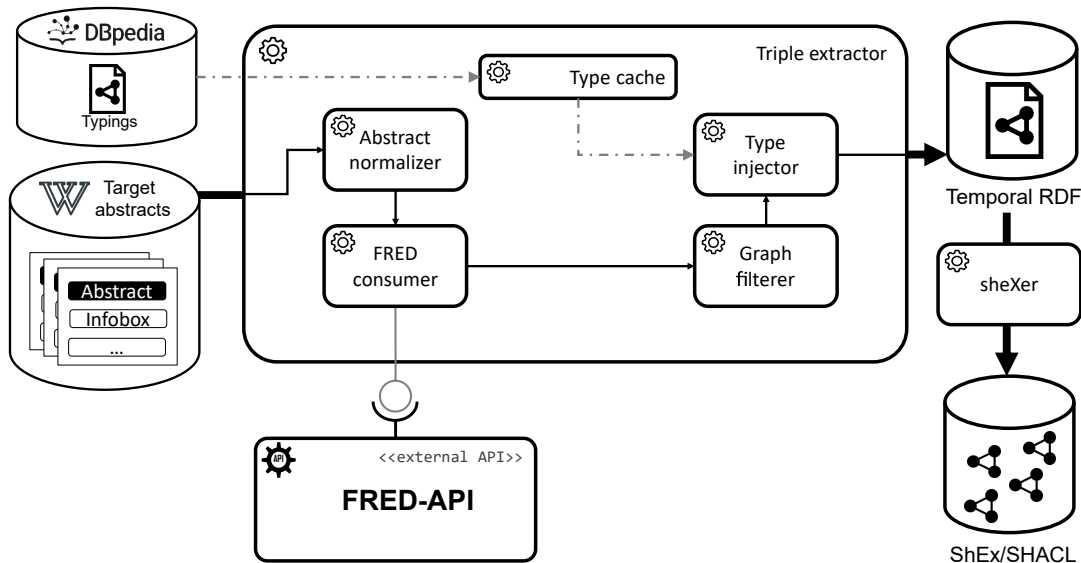
1. The input text is transformed to Discourse Representation Structures (DRSs), based on Discourse Representation Theory [291]. DRSs are informally called *boxes* due to its box-like visual representation. For example, a sequence of sentences such as “Larry likes coffee. If someone likes it, he drinks it.” can be represented using the DRS shown in Figure 5.4. DRS supports First Order Logic. Using the previous example, it could be inferred that “Larry drinks coffee”. In order to transform natural language into DRSs, FRED uses the tool Boxer [292].
2. Boxer is able to perform some Coreference Resolution (CRR) tasks. However, the FRED system improves those capacities by integrating Boxer with other tools. Pronoun CRR is improved by computing Boxer’s output with CoreNLP [293]. Similarly, Boxer’s NER capabilities are improved by computing its output with TAGME [294]. That system is able to recognize entities in short texts. It also performs EL, as it can link the recognized entities to the title of a Wikipedia page. Boxer, CoreNLP, and TAGME are combined to produce the DRSs that will be processed in the next stages.
3. The DRSs are sent to a module to be transformed into triples. That module performs several heuristic-based subprocesses to achieve such a general goal:
 - **Taxonomy induction.** FRED is able to identify compound terms and infer taxonomies. For example, FRED may find that the event of a class is a *SoccerMatch*. This is a compound term formed by the ideas *Soccer* and *Match*. FRED can express this notion by producing a triple such as (*fred:SoccerMatch* , *rdfs:subClassOf* , *fred:Match*).

⁵RDF basic model is binary, as a triple represents a relations between two elements. N-ary relations are used to link *n* elements. Extra vocabularies can be defined in order to support semantically rich properties to represent n-ay relations.

- **Variable reification.** FRED represents n-ary relations using RDF reification schemata.
 - **Periphrastic relation extraction.** FRED recognizes periphrastic relations annotated by means of prepositions, such as “born to” or “going to”. Using prepositions as relations in a semantic graph can lead to incongruence. For example, the notion of “to” in “born to” and in “going to” is quite dissimilar. FRED identifies the element associated to a certain preposition to create periphrastic relations, such as *fred:bornTo* and *fred:goTo*.
 - **Frame and situation extraction with Semantic Role Labeling (SRL).** The *n*-ary relations are represented by means of frames or verbs from existing vocabularies when possible. Specifically, FRED reuses verbs as frames from VerbNet [295] and FrameNet [207]. Roles w.r.t. the frame situation are identified and labeled too.
 - **Representation of several aspects,** such as role propagation, entailment, modality, negation, qualities, and tense.
4. The heuristic-based triplication outputs a first version of an RDF graph. This preliminary result is sent to a module of graph enrichment which executes the following tasks to produce a final result for the user:
- **Entity Linking.** TAGME’s results are used in the graph enrichment module to perform EL. Entities recognized by TAGME are linked to a DBpedia URI using *owl:sameAs* axioms. Both individuals and classes can be enriched with these axioms.
 - **Type induction.** The *owl:sameAs* axioms can produce new *rdf:type* declarations by using the semantics declared in external ontologies.
 - **Ontology alignment via Word Sense Disambiguation (WSD).** FRED performs WSD by linking classes with equivalent or broader elements in WordNet [206] and BabelNet [296] when appropriate. The WSD process can also perform alignments with WordNet *supersenses*⁶ and a subset of DOLCE+DnS Ultra Lite (DUL) classes. This process is performed by means of the UKB tool [297].
 - **Compositional Association Induction.** The taxonomies induced from compound terms are enriched in this stage. When the feature that it is used to specialize a class is a noun, FRED adds triples with the property *dul:associatedWith* to the results. When this feature is expressed as an adjective or adverb, FRED uses *dul:hasQuality*. For example, *fred:SoccerMatch* is a specialization of *fred:Match* that is related with the concept of *Soccer* (a noun). In this case, FRED would enrich the graph by adding the triple $(fred:SoccerMatch , dul:associatedWith , fred:Match)$.
 - **Validation and possibly correction** of the RDF content produced after executing every heuristic for triple extraction and graph enrichment.
 - **Textual annotation.** FRED can annotate pieces of the text parsed with the triples produced. This annotation is performed using the Earmark vocabulary [298] and the NLP Interchange Format (NIF) [299].

⁶WordNet senses are sometimes too fine-grained even for human annotators. WordNet *supersenses* consist of a much smaller coarse-grained set of elements, with less precise semantics but easier to handle for classification tasks.

FIGURE 5.5: SEITMA-F: implementation based on FRED



5. The graph produced during the enrichment stage is returned as result.

The processes described up to now works with English content, but FRED accepts 48 different input languages. Non-English content is translated to English using external APIs, and then sent to FRED's core workflow.

FRED can be executed in several ways. One can use a public on-line demo⁷ that consumes a REST⁸ API. FRED is also offered as a Python library⁹ that consumes the same REST API. This REST API can be accessed by some other mechanisms such as [CURL](#) commands.

At the time of this writing, anyone can use the public on-line demo. However, in order to work with the Python library or to consume the REST API by any other means, one need to request an API key to FRED's maintainers¹⁰.

5.3.2.2 SEITMA-F implementation

The internal architecture of SEITMA-F's triple extractor is detailed in Figure 5.5. SEITMA-F's triple extractor integrates FRED in a workflow composing the following steps:

1. The system receives some raw Wikipedia abstracts. Those abstracts are written in markdown [300], as they are found in the Wikipedia XML dumps.
2. The abstracts are sent to a *Abstract normalizer* module. This module removes markdown features to transform the content in plain text.
3. The normalized content is sent to the *FRED consumer* module. The main goal of this module is to query the FRED API to transform the input text into an RDF graph. To do so, it performs several actions:

⁷<http://wit.istc.cnr.it/stlab-tools/fred/demo/> Accessed in 2022/05/03.

⁸REST stands for Representational State Transfer.

⁹<http://wit.istc.cnr.it/stlab-tools/fred/fredlib.py> Accessed in 2022/05/03.

¹⁰Cf. <http://wit.istc.cnr.it/stlab-tools/fred/demo/> Accessed in 2022/05/03.

- It splits the abstract in smaller units (chunks), usually composed by a couple of sentences. These units are sent to the FRED API in different requests. Splitting the abstracts with this strategy can lead to CRR errors due to a lack of context for some sentences. However, the FRED API does not support the computation of big text chunks. The splitting is performed in such a way that FRED can process each unit within a single API request.
 - It handles API response errors to retry the computation of conflictive units by dividing them in smaller chunks.
 - As aforementioned, one needs an API Key to work with FRED API. Each API key is associated with some usage restrictions w.r.t. maximum petitions within a minute and maximum petitions within a day. The FRED consumer module schedules requests to FRED API so the API key limits are not exceeded.
 - It integrates the results of successive API calls into a single RDF graph result.
4. The results collected by the FRED consumer are sent to the *Graph filterer* module. Even if this content is always related somehow to the central abstract's entity, the triples extracted are not always directly linked with the abstract's target entity. The mission of the Graph filterer is to locate and adapt those triples which are directly linked with the abstract's entity. This module works as follows:
- (a) For a certain graph G_i obtained from the abstract of an entity e_i , the module looks for the DBpedia URI of e_i in G_i . Finding e_i 's DBpedia URI in G_i implies that FRED has correctly identified a mention of e_i in the text, as it has been able to enrich it with the adequate DBpedia URI. If this URI is not found, G_i is discarded, as there is no evidence that the knowledge produced by FRED is actually referring to e_i .
 - (b) The module looks for every node n_{i_j} such that a triple $(n_{i_j}, owl:sameAs, e_i)$ exists in G_i . n_{i_j} nodes are expected to be URIs in the *fred* namespace.
 - (c) A temporal graph G'_i is created. This graph is seed with every triple of G_i using a n_{i_j} node. However, these triples are stored in G'_i changing every mention of a n_{i_j} node by the DBpedia URI of e_i .
 - (d) The triples containing types of those elements found in the opposite position of each triple using a n_{i_j} are also added to G'_i .
 - (e) The module removes from G'_i any triple whose predicate belong to a user-configured list L_{ban} . We have introduced L_{ban} to avoid some properties frequently found but barely meaningful, such as prepositions (*fred:to*, *fred:as*, etc.)¹¹ that were not used to create a compositional relation within FRED's workflow.
 - (f) The content produced by the Graph filterer is sent to the *Type injector* module. Here, every DBpedia URI is located. For each URI w_i , the module introduces every type declared for w_i within the DBpedia ontology namespace which was not already include by FRED in the resulting graph. For

¹¹In FRED's output, the prepositions *to* and *as* can be used as predicate in a triple and are represented with the URIs `<http://www.ontologydesignpatterns.org/ont/fred/domain.owl#to>` and `<http://www.ontologydesignpatterns.org/ont/fred/domain.owl#as>` respectively.

such a task, it uses a *Type Cache*. The Type Cache module is built using the same data and works in the same manner as described for SEITMA-L.

- (g) Every Graph G_i produced from the abstract of an entity e_i is integrated in a single RDF graph G . G is the output sent to the shape extractor.

Shapes from triples SEITMA-F also uses sheXer with `inverse_paths = True` and `all_classes_mode = True` to extract shapes from the triples generated.

The temporal RDF graphs generated by SEITMA-F are centered in the content related to the target entities, i.e., entities that are an instance of a target class. Whenever the system is able to extract knowledge of at least one instance of a target class, a shape for this target class is produced. During this process, target entities can be linked with non-target entities. The types associated with those target entities using `rdf:type` triples also produce a shape.

Usually, when SEITMA-L and SEITMA-F are fed with the same target abstracts, SEITMA-F is able to extract more shapes. This is because SEITMA-L works exclusively with classes within the DBpedia ontology, while SEITMA-F graphs also uses other types generated by FRED.

About input files and system requirements - SEITMA-F requires the following inputs:

- **Wikipedia dump file** containing the target abstracts to mine.
- **DBpedia typings**, to build the Type cache.
- A valid **FRED API key**.

Every SEITMA-F computation is performed locally with the exception of the requests to FRED API. Those requests can be a bottleneck though. The API Key that we used during our experiments allows us to perform a maximum of 7,500 calls per day. Since the API calls to FRED API usually contain two sentences, this makes a maximum of 15,000 sentences parsed per day in optimal conditions. However, the number of sentences parsed per day is usually lower due to sentences that are too long to be parsed in couples and API calls that return an error.

SEITMA-F needs less resources than SEITMA-L to allocate caches in main memory, as SEITMA-F uses a Type cache but does not need a Back-link cache.

5.4 Experiments

In this section, we describe some experiments with our two SEITMA implementations. Those experiments are not designed to produce optimal shapes for each abstract parsed, but to demonstrate the potential of systems based on SEITMA's proposals. Actually, the notion of *optimal* for the task of extracting shapes from natural language pieces can be hard to define without a benchmark to compare with or, at least, a well-established context of application.

The target abstracts to extract shapes are chosen using importance criteria based on ClassRank and PageRank notions. The shapes extracted from those abstracts let us discuss the strengths and weaknesses of each prototype, propose application domains, and suggest different configurations, adaptations, or new prototype proposals.

In order to make our experiments reproducible, we provide a link to every file, configuration, and material used in this process¹². The Wikipedia dump used is available on-line on our data servers¹³. The rest of input files are related to DBpedia. They are all available in a DBpedia’s Databus collection or our data servers¹⁴. That Databus collection includes:

- **Object to object relations:** files *mappingbased-objects* and *infobox-properties*.
- **Links between Wikipedia pages:** file *wikilinks*.
- **Instance-class relations:** file *instance-types*.

We have extracted shapes for 200 target classes, which are the 200 most important classes of the DBpedia ontology according to ClassRank (cf. chapter 3 for more information about ClassRank). ClassRank was executed using *rdf:type* as the only class-pointer and setting the standard configuration $\alpha = 0.85$. The set of classes C to rank consists of every element in DBpedia’s ontology within the namespace *dbo*. Since C is known a priori, it was not necessary to execute the ClassRank class-discovery stage detailed in Algorithm 1 of Chapter 3.

We have detected the most important instances of each class using the PageRank algorithm [301]. PageRank was executed with the standard configuration $\alpha = 0.85$ over an object-to object DBpedia subgraph. This subgraph was obtained by merging the object-to-object triples in the *mappingbased-objects* and the *infobox-properties* files. This means that we have ranked the importance of instances w.r.t. the DBpedia link structure. A different and feasible option would have been to rank entities w.r.t. to the Wikipedia’s structure. For that, one can apply PageRank over the wikilinks graph, which can be easily parsed using DBpedia’s wikilinks file.

The maximum number of instances used per class was set to 300 items. Nevertheless, note that not every class on the top 200 ranking has 300 instances. Every available instance among that group of 300 (as most) was used to get a shape for their corresponding class.

There are some special classes whose instances are relevant for the DBpedia graph but do not have associated Wikipedia pages. An insightful example of such a class is *dbo:CareerStation*¹⁵. This class ranks 2nd with ClassRank, but none of its instances is related to a Wikipedia page. Those kinds of special instances were excluded from further analyses. Consequently, some of the classes of the top 200, such as *dbo:CareerStation*, do not produce any shape as they cannot be mined in Wikipedia using any SEITMA implementation.

Thus, the total number of abstracts processed was 38,247. Note that some of those abstracts are linked to entities which are among the 300 most important instances of more than one target class.

¹²In order to reproduce SEITMA-F experiments, one should use a personal FRED API key

¹³The dump used is no longer available in Wikipedia servers, but it can be downloaded from the following link: <http://data.weso.es/setima/>. This data is shared using GDFL license, i.e., under the same conditions of Wikimedia’s servers. Check license description at <https://www.gnu.org/licenses/fdl-1.3.html> Accessed in 2022/05/03.

¹⁴The Databus collection contains most of the target DBpedia files and can be downloaded at https://databus.dbpedia.org/danifdezalvarez/collections/seitma_inputs. The OWL file containing class and property definitions can be downloaded from https://data.weso.es/seitma/dbpedia_2021_07.owl Accessed in 2022/05/03.

¹⁵The property *dbo:CareerStation* “links to a step in the career of a person, e.g. a soccer player, holding information on the time span, matches and goals he or she achieved at a club”. Cf. <http://dbpedia.org/ontology/careerStation> Accessed in 2022/05/03.

One can obtain the list of top classes and their most important instances by executing ClassRank and PageRank as it has been indicated. However, to facilitate the reproducibility of our experiments, we have published the list of top classes and their respective instances as a JSON file¹⁶.

We executed SEITMA-L and SEITMA-F using the target abstracts and input files described. In the following subsections, we detail the configurations and results obtained with each prototype.

5.4.1 Experiments with SEITMA-L

We have divided the content related to SEITMA-L experiments in three sections. First, we explain the inputs and settings used. Then, we analyze the output obtained. Finally, we discuss the prototype’s potential and applicability to other scenarios.

5.4.1.1 Inputs and settings

As already explained, SEITMA-L requires two types of input abstracts. On the one hand, it needs some reference abstracts A_r to extract positive and negative examples. These examples are used to train $k \cdot 2$ classifiers, where k is the number of properties used in those examples. On the other hand, it requires some target abstracts A_t to extract triples using the trained classifiers.

Let us denote F_{A_r} the set of examples extracted from A_r . Also, let us denote F_{A_t} the set of candidates extracted from A_t . In this experiment, $A_r = A_t$. Despite this, F_{A_r} and F_{A_t} are disjoint, i.e., $F_{A_r} \cap F_{A_t} = \emptyset$. Also, it can be observed in the data collected that $|F_{A_r}| \ll |F_{A_t}|$. There are several reasons for those facts:

- At the stage of training data generation, negative examples of a certain property r in a certain abstract a are generated just in case there is a positive example of r in a . Since this is not the case at the stage of candidate generation, the number of candidates for predictions tends to be much bigger than the number of training examples. Using the abstracts described, the original set of candidates C detected contains 2,414,222 elements, while number of examples F_{A_r} collected is 11,338.
- Domain and range inference are used to find negative examples, but are not required to find positive ones. For a certain abstract a_e of an entity e , we consider that a mention w_i is a positive example when there is a triple t_{e,w_i} in DBpedia that links w_i and e by means of a relation r . The existence of t_{e,w_i} is considered enough evidence to determine that the context of w_i is a trustworthy positive example of the relation r . However, on some occasions, t_{e,w_i} produces some constraint violation according to the domain and range definition of r in the DBpedia ontology. Therefore, w_i is used as a positive example in the training data stage, but it is not even detected as candidate in the prediction stage. 3,872 examples do not belong to the original set of candidates C , i.e., 21% of the elements in F_{A_r} .

Just 7,466 of the elements of C are in F_{A_r} too. We denote C' the set of candidates that exclude those elements, i.e. $C' = C \setminus F_{A_r}$. In our experimentation, C' is used as the set of candidates F_{A_t} . Then, F_{A_t} and the set of examples F_{A_r} totally disjoint. In those conditions, every triple predicted in SEITMA-L’s triple generation stage is

¹⁶One can download this file using the following URL: http://data.weso.es/seitma/300instances_from_200classes.json Accessed in 2022/05/03.

new, i.e., it is a triple which is not in DBpedia. The total number of elements in F_{A_t} is 2,406,756. In order to facilitate the reproduction of this experiment, we have published the list of final examples and final candidates used. They are contained in two Comma Separated Values (CSV) files. In those files, each row represents an example/candidate already mapped to a set of features by SEITMA-L¹⁷.

Despite LAREWA's approach for triple prediction is language-agnostic, the abstracts used for training and the abstracts used for predictions should be in the same language. This is because both syntactical and cultural aspects may affect how the information is expressed, or even what kind of information is more frequently added to Wikipedia.

Our intuition is that a similar premise could affect the performance of the models if they are trained with central entities but used to predict knowledge in non-central ones. That is, the most central entities of each class may be the ones which are more actively edited and moderated by the community, so they could contain more and better written knowledge than non-central ones. Such special features of central entities may decrease the performance of prediction models when they are used with less important elements. For this reason, we decided to use the same abstracts to train data models and to produce triples from.

SEITMA-L needs a value for three internal thresholds:

- θ_s , sheXer's acceptance threshold (cf. section 4.4). We arbitrarily set $\theta_s = 0$. As this experimentation seeks to analyze the potential of SEITMA-L, this value allows us to inspect every piece of knowledge extracted.
- θ_p , minimum precision that a model must achieve to not be discarded. We arbitrarily set $\theta_p = 0.75$. This is a low value for the specific task of triple prediction. However, SEITMA-L aims to produce shapes rather than triples. A moderated error on triple prediction in our scenario can affect the sheXer's constraint ratios, and thus alter the order of constraints, but it is not expected to cause the removal of a constraint, especially when $\theta_s = 0$.
- θ_n , minimum number of examples to train a model. We chose a value for θ_n that let us keep at least the arbitrary percentage of 30% of all the models with at least a training example (achieving a minimum precision $\theta_p = 0.75$). Therefore, we set $\theta_n = 13$, as using such a value 66 out of 220 potential models can be accepted. $\theta_n = 13$ can be a low number too, as we may be accepting models trained with non-representative enough data. However, such a setting for θ_n let us produce shapes using properties which are not frequently observed among the examples in this experimental proof of concept, even if those models may have over fitting problems due to the sample size [302].

In Table 5.1, we provide summary data about the candidate and accepted models in our experiments.

Let us denote P-R a combination of a Property p and a certain Role r (one of subject or object) for an abstract's entity in a triple. Each P-R is a candidate to train a classifier. As one can see in Table 5.1, the total number of P-R seen in the training abstracts is 220. However, despite using a low minimum sample size $\theta_n = 13$, only 88 P-R had enough examples to train and evaluate a model. Using a minimum target precision of $\theta_p = 0.75$, only 66 models were accepted. With this setting, the total

¹⁷Examples can be downloaded at http://data.weso.es/seitma/seitma_l_example_features.zip. Candidates can be downloaded at http://data.weso.es/seitma/seitma_l_candidate_features.zip. Accessed in 2022/05/03.

N° of P-R with at least a positive example	220
N° of P-R considered with $\theta_n = 13$	88
N° of models accepted with $\theta_n = 13$ and $\theta_p = 0.75$	66
N° of properties with at least an accepted model	54
Average precision of accepted models	0.93
Average sample size of accepted models	268.52
Standard deviation w.r.t. precision of accepted models	0.07
Standard deviation w.r.t. sample size of accepted models	316.98

TABLE 5.1: Summary data about automatic classifiers in SEITMA-L experiments.

number of properties used to produce triples is 54. That is, some properties generated two models, so they can be used to predict direct and inverse triples w.r.t. the role of the abstract’s entity.

The low values of θ_n and θ_p were set to accept some interesting domain specific properties with few uses. Note that, despite those low threshold values, the average precision of the used models is 0.93, while the average size of the sample for training a P-R model is 316.98. As the standard deviation w.r.t. precision indicates, most of the models are close to the 0.93 average precision. However, we can observe a huge dispersion w.r.t. sample size. Several P-R classifiers were trained using small samples, while some others had much more training examples than the average.

5.4.1.2 Outputs

In this subsection, we first show an example of a shape extracted with SEITMA-L. We comment some features that can be observed in this example and let us perform a first qualitative analysis of SEITMA-L’s output. Then, we perform a statistical analysis of the shapes generated with SEITMA-L.

Example shape - In Figure 5.6, we show a reduced version of the shape `:Band` produced by SEITMA-L during our experiments. The original shape `:Band` is 109 lines long. Part of the content has been removed so the shape can be displayed in this document (comments starting with “# ...” mark removed parts). This shape aims to generalize features observed among the instances of the class `dbo:Band`.

First, note that there is not a single Triple Constraint (TC)¹⁸ whose node constraint is a literal type. This is true for every shape generated with SEITMA-L. The SEITMA-L’s triple generator module mines Wikipedia object-to-object relations. No object-to-literal relation is computed during that stage, so no triple nor shape containing literals is produced.

Note that many `:Band`’s TCs define a cardinality that includes a zero-case. Most of the times, this cardinality is *zero-to-many* (“*”), but examples of optional (“?”) are also found. This fact is not just observed in the shape `:Band`, but also in many shapes produced by SEITMA-L, especially those built using a large number of triples.

This fact indicates that few features mined by sheXer are observed in every instance of a certain class. When mining a big crowd-sourced dataset such as Wikipedia, this observation makes sense in general. It is unlikely that every instance of a certain class defines a value for a certain property.

¹⁸TC is a non-standard abbreviation of the concept *triple constraint*. We will use it to improve the readability of this chapter.

FIGURE 5.6: `:Band`. A shortened example of a shape produced by SEITMA-L

```

1  :Band
2  {
3
4  rdf:type [ dbo:Band ] ; # 100.0 %
5  dbo:designer IRI * ;
6  # 59.34065934065934 % obj: IRI. Cardinality: +
7  # ...
8  # 56.043956043956044 % obj: @:MusicalArtist. Cardinality: +
9  # 2.197802197802198 % obj: @:Artist. Cardinality: {1}
10 # 1.098901098901099 % obj: @:Royalty. Cardinality: {1}
11 # 1.098901098901099 % obj: @:Guitarist. Cardinality: {1}
12 # 1.098901098901099 % obj: @:Politician. Cardinality: {1}
13 # 1.098901098901099 % obj: @:ComicsCreator. Cardinality: {1}
14 dbo:location IRI * ;
15 # 45.05494505494506 % obj: IRI. Cardinality: +
16 # ...
17 # 34.065934065934066 % obj: @:City. Cardinality: +
18 # 7.6923076923076925 % obj: @:Settlement. Cardinality: {1}
19 # 4.395604395604396 % obj: @:AdministrativeRegion...
20 # 3.296703296703297 % obj: @:Town. Cardinality: {1}
21 # 3.296703296703297 % obj: @:Country. Cardinality: {1}
22 # ...
23 dbo:origin IRI * ;
24 # 45.05494505494506 % obj: IRI. Cardinality: +
25 # ...
26 # 34.065934065934066 % obj: @:City. Cardinality: +
27 # 6.593406593406594 % obj: @:Settlement. Cardinality: {1}
28 # 4.395604395604396 % obj: @:AdministrativeRegion...
29 # 3.296703296703297 % obj: @:Town. Cardinality: {1}
30 # 3.296703296703297 % obj: @:Country. Cardinality: {1}
31 # 1.098901098901099 % obj: @:CityDistrict. Cardinality: {1}
32 # ...
33 ~ <http://dbpedia.org/ontology/developer> IRI * ;
34 # 37.362637362637365 % obj: IRI. Cardinality: +
35 # 31.868131868131865 % obj: @:MusicGenre. Cardinality: +
36 # ...
37 # 15.384615384615385 % obj: @:Song. Cardinality: +
38 # 14.285714285714285 % obj: @:Album. Cardinality: +
39 # 12.087912087912088 % obj: @:Person. Cardinality: +
40 # ...
41 dbo:owningCompany @:RecordLabel * ;
42 # 23.076923076923077 % obj: @:RecordLabel. Cardinality: +
43 # ...
44 dbo:hometown IRI ? ;
45 # 8.791208791208792 % obj: IRI. Cardinality: {1}
46 # 7.6923076923076925 % obj: @:City. Cardinality: {1}
47 # 1.098901098901099 % obj: @:Town. Cardinality: {1}
48 dbo:associatedMusicalArtist @:Guitarist ? ;
49 # 1.098901098901099 % obj: @:Guitarist. Cardinality: {1}
50 ~ dbo:publisher @:Album ?
51 # 1.098901098901099 % obj: @:Album. Cardinality: {1}
52 }
53

```

There are different situations in which a TC with a zero-case cardinality can be generated:

- The feature studied is correctly identified as optional. For instance, the rights of a certain band may or may not be owned by a company.
- The feature is not optional, but the source data is not complete. For example, the shape indicates that 45% of the bands has an origin. This may be true according to the mined information, but it is probable that this happens due to incompleteness of the source of information.
- The approach fails to extract the target knowledge. This shape indicates that only 1% of the bands has some `dbo:associatedMusicalArtist`. It also indicates that, when this association exists, the artists is a `:Guitarist`. Both statements seem inaccurate. It is probable that more bands are associated to musical artists in Wikipedia, and that those artists have roles different than guitarist.

The actual reason of a cardinality including a zero-case may be hard to automatically identify, and to fix in case it is an error. Wrong zero-case cardinalities, regardless of their cause, could be fixed with human supervision a posteriori, or by adding some extra computation layers to the shape extractor module. For example, such layers could consist of using external and trustworthy ontological knowledge about the target classes, or implementing some ML approach able to identify non-optional relations.

A salient exception of this observation w.r.t. TC cardinalities are the typing constraints. Every shape must have at least a TC which uses a value set to indicate the type that is related with the shape label. This type is the feature that sheXer uses to distinguish which instances are used to extract which shapes. Then, TCs describing the main type of a shape have a cardinality of exactly one and are observed in 100% of a shape's related instances.

Another feature observed is that the node constraint of most of the `:Band`'s TCs is the macro `IRI`. This happens because there is not a unique shape that conforms with every node observed at the opposite side of the focus node for a certain relation. This fact causes the TC to be less informative for description tasks and less useful for validation tasks, as the combination of the node constraint `IRI` and the `'*` cardinality may be too unspecific.

Nevertheless, sheXer provides text comments with statistical observations that can help to increase the value of each constraint. For example, a node conforming with `:Band` could have a `dbo:location` of type `IRI`. The comments indicate that 45% of them has at least one location though. Also, several comments indicate frequent types of those locations, such as `:City`, `:Settlement`, `:AdministrativeRegion`, `:Town`, or `:Country`. In addition, several comments indicate that a frequent cardinality for this relation is exactly one. All these pieces of information can help human experts or further processing layers to transform too general TCs into more specific values regarding the specificity of the node constraint and the cardinality.

In the `:Band` shape, we can see some examples of TCs with a node constraint different to `IRI`. In general, there are two main scenarios where such precise constraints tend to appear:

- **Properties with precise domain and range definitions.** Domain and range definitions tend to use classes general enough to not cause constraint violations. For example, the range defined for the property `dbo:origin` is

`dbo:PopulatedPlace`¹⁹, and `dbo:PopulatedPlace` is a class with many different subclasses. This makes unlikely to happen that each `dbo:origin` of each `:Band` is an instance of the same subclass. To increase the node constraint specificity, it is feasible to think about using a *supershapes* representing the essential features of `dbo:PopulatedPlace`. This supershape could be extended by shapes linked to the subclasses of `dbo:PopulatedPlace`. However, at the time of this writing, the concept of *shape inheritance* is not yet supported by ShEx nor SHACL²⁰.

In contrast, the range of `dbo:owningCompany` is `dbo:Company`²¹, which is a superclass of `dbo:RecordLabel`. Compared to `dbo:PopulatedPlace`, the class `dbo:Company` restricts more the possible object types. SEITMA-L is benefited from this at the triple generation stage, as the number of candidates for this relation is reduced. Then, the shape generator module works with triples whose objects are more homogeneous, and the chances of finding precise node constraints are increased.

- **Properties rarely observed.** This seems the case of the TCs expressed in lines 48 and 51. Those two features are just observed for 1% of the `:Band` instances. When a certain feature is observed for few instances, the chances of agreement w.r.t specific node constraint are increased.

The `:Band` shape also includes some comments that seem to reveal wrong predictions in the triple generation stage. For example, line 39 indicates that it has been observed that 12% of the bands are developers of one or more `:Person` nodes.

A triple such as $(b, \text{dbo:developer}, p)$, where b is an instance of `dbo:Band` and p is an instance of `dbo:Person` is, indeed, ontologically correct: `dbo:developer` defines a range of `dbo:Agent`, which is a superclass of `dbo:Person`, and it does not define any domain. However, the property `dbo:developer` is described in its ontology in the following terms: “Developer of a Work (Artwork, Book, Software) or Building (Hotel, Skyscraper)”²². This definition does not seem to suggest that an instance of `dbo:Person` is a valid object for a triple whose predicate is a `dbo:developer`. Also, one can assume that 12% of the Wikipedia abstracts of `dbo:Band` instances (maybe not even a single one) state such a relation between a person and a band.

In this example, it is also noticeable that the actual number of TCs (7 direct and 2 inverse) is low compared with the potential notions that a human could have annotated from the target sample of 300 bands. In figure 5.7, we show a subsection of the Metallica’s abstract²³, which is one of the target abstracts used to extract the shape `:Band`. In this Figure, we have highlighted in different colors some property-value pairs that a human annotator could have extracted from the abstract. Those pairs contain some values which do appear in the shape `:Band`, such as the notions of *hometown* (Los Angeles), or *publisher* (of several albums). However, there are some others that have not be included in the obtained shape `:Band`, such as *band members*, *genre*, or *award nominations*.

¹⁹Check range definition at <https://dbpedia.org/ontology/origin> Accessed in 2022/05/03.

²⁰As shown in section 2.6, both languages have mechanisms to re-use the constraints of a shape s_1 in the definition of another shape s_2 . However, the semantics of those mechanisms does not imply that s_2 is a s_1 , which is a core idea of inheritance in programming languages.

²¹<https://dbpedia.org/ontology/owningCompany> Accessed in 2022/05/03.

²²<https://dbpedia.org/ontology/developer> Accessed in 2022/05/03.

²³<https://en.wikipedia.org/wiki/Metallica> Accessed in 05/04/2022.

FIGURE 5.7: Part of Metallica’s Wikipedia page.

Metallica

From Wikipedia, the free encyclopedia

This article is about the band. For its eponymous fifth album, see [Metallica \(album\)](#). For other uses, see [Metallica](#)

Metallica is an American heavy metal band. The band was formed in 1981 in Los Angeles by vocalist/guitarist James Hetfield and drummer Lars Ulrich, and has been based in San Francisco for most of its career.^{[1][2]} The band's fast tempos, instrumentals and aggressive musicianship made them one of the founding "big four" bands of thrash metal, alongside Megadeth, Anthrax and Slayer. Metallica's current lineup comprises founding members and primary songwriters Hetfield and Ulrich, longtime lead guitarist Kirk Hammett and bassist Robert Trujillo. Guitarist Dave Mustaine (who went on to form Megadeth after being fired from the band) and bassists Ron McGovney, Cliff Burton and Jason Newsted are former members of the band.

Metallica first found commercial success with the release of its third album, *Master of Puppets* (1986), often cited as one of the heaviest and most influential thrash metal albums^[citation needed]. The band's next album, ...*And Justice for All* (1988), gave Metallica its first Grammy Award nomination. Its eponymous fifth album, *Metallica* (1991), the band's first not to root predominantly in thrash metal, appealed to a more mainstream audience, achieving substantial commercial success and selling over 16 million copies in the United States to date, making it the best-selling album of the SoundScan era. After experimenting with different genres and directions in subsequent releases, Metallica returned to its thrash metal roots with the release of its ninth album, *Death Magnetic* (2008), which drew similar praise to that of the band's earlier albums. The band's most recent album is *Hardwired... to Self-Destruct*, released in 2016.

The mentioned properties are not represented in the shape because 1) the knowledge that they represent does not exist in DBpedia, or 2) the models trained to predict such property were not reliable enough to make triple predictions.

Shapes’ stats - All the shapes obtained with SEITMA-L in this experiment are public and can be downloaded²⁴. We have summarized the content of those shapes in several tables and charts. In Table 5.2, we describe general information, such as number of shapes, TCs, and comments produced. We also show average values and standard deviations at shape, TC, and comment level. Several conclusions can be extracted from this table.

First, note that it is a coincidence that the number of extracted shapes is exactly the number of target classes. Some of the target classes do not have instances or SEITMA-L has not been able to extract triples from their instances. At the same time, some elements linked with the target entities has a type which is not part of the initial target 200 classes.

The average number of constraints per shape is 6.08, with a standard deviation of 3.47. Those facts allow us to generalize some of the stated observations about the shape *:Band*. In general, the shapes produced by SEITMA-L do not contain a high number of constraints compared to the number of potential relations that a human annotator could have extracted from an abstract. This is frequently caused by the limited amount of positive and negative examples that can be automatically extracted from an abstract.

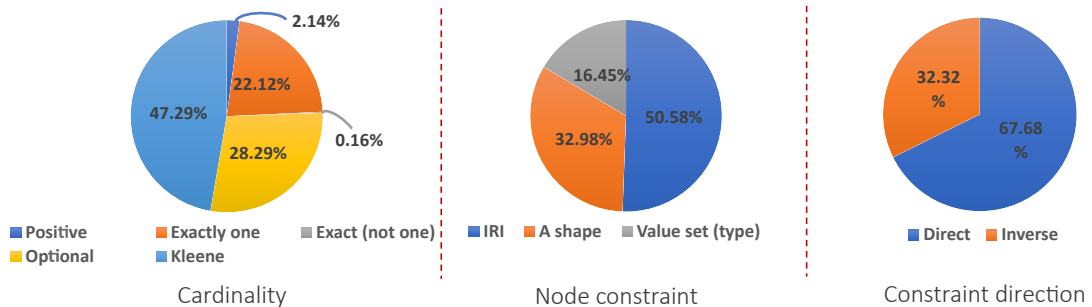
The average number of comments associated to a shape is 33.78. We can observe a great dispersion between shapes w.r.t. this feature, as the standard deviation is 34.04. The average number of comments associated to a certain constraint is 5.56.

²⁴One can download the shapes at http://data.weso.es/seitma/seitma_l_shapes.zip. The intermediate triples generated to extract those shapes are public too and can be downloaded at http://data.weso.es/seitma/seitma_l_triples.zip Accessed in 2022/05/03.

Number of shapes	200
Number of TCs	1216
Number of comments	6756
Avg. constraints per shape	6.08
St.dev. constraints per shape	3.47
Top n° of constraints in a shape	18
Avg. comments per shape	33.78
Std.dev. comments per shape	34.04
Avg. comments per constraint	5.56
Std.dev. comments per constraint	5.00
Avg. ratio constraints	43.75
Avg. ratio comments	11.34

TABLE 5.2: Statistics about shapes in SEITMA-L results

FIGURE 5.8: Cardinality, node constraint, and constraint direction w.r.t. focus node of TCs in SEITMA-L results.



We can also observe a noticeable dispersion on this data, as the standard deviation is 5.00.

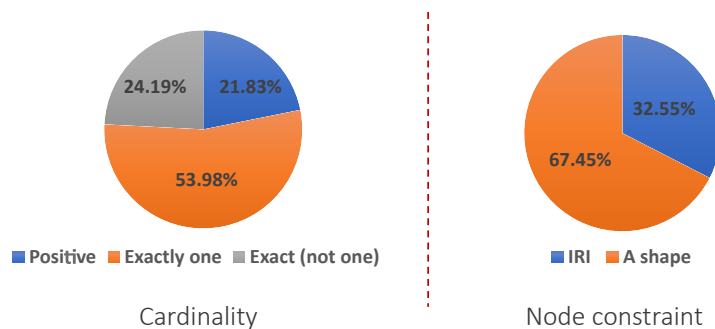
The extremes w.r.t. number of comments per shape is mainly caused by the semantics of the properties that generated a reliable model. Some classes use those properties frequently, while some other barely use them. Classes with many uses for a property generated many triples. Then, the chances of finding specific cardinalities or node constraints among those examples increases. In contrast, few examples lead to situations in which sheXer cannot find many different patterns among the target triples. A reduced number of instances for a certain class also causes to have less associated comments, as it also decreases the target sample that sheXer can use to mine a certain relation.

In Figure 5.8, we show the distribution of TCs w.r.t. cardinality, path sense (direct or inverse w.r.t. the focus node), and node constraint. In Figure 5.9, we provide similar information w.r.t. text comments.

The trends observed in the shape *:Band* w.r.t. cardinality seems to apply to the rest of the shapes. The two most frequent cardinalities are '*' and '?', i.e., those including the zero-case. There is a noticeable proportion of constraints with cardinality '1' too. 200 of those constraints (74% of the constraints with cardinality '1') are type specifications. The rest of them are mostly observed in shapes built using a small number of target instances. Cardinalities which are not '1' nor include the zero case are marginal and always observed in shapes built using few target nodes.

The trends observed in *:Band* w.r.t. node constraint distribution are also confirmed. 50.58% of the node constraints are the macro IRI. 16.45% correspond to

FIGURE 5.9: Cardinality and node constraint of comments in SEITMA-L results.



value sets used for type specifications, and 32.98% use another shape label. This last type of node constraint appears more frequently in TCs with a low trustworthy score or built using a small number of target instances.

One can observe that the number of direct TCs is approximately twice the size of inverse ones. This is a proportion similar to what we observed in the *:Band* shape. We think that this occurs because each abstract is focused in a given entity and, in RDF, when one want to state something about an entity e , e is naturally used as subject instead of object.

As one can see comparing Figure 5.8 and Figure 5.9, the distribution of node constraints and cardinalities among the comments differs from the distributions observed among the TCs.

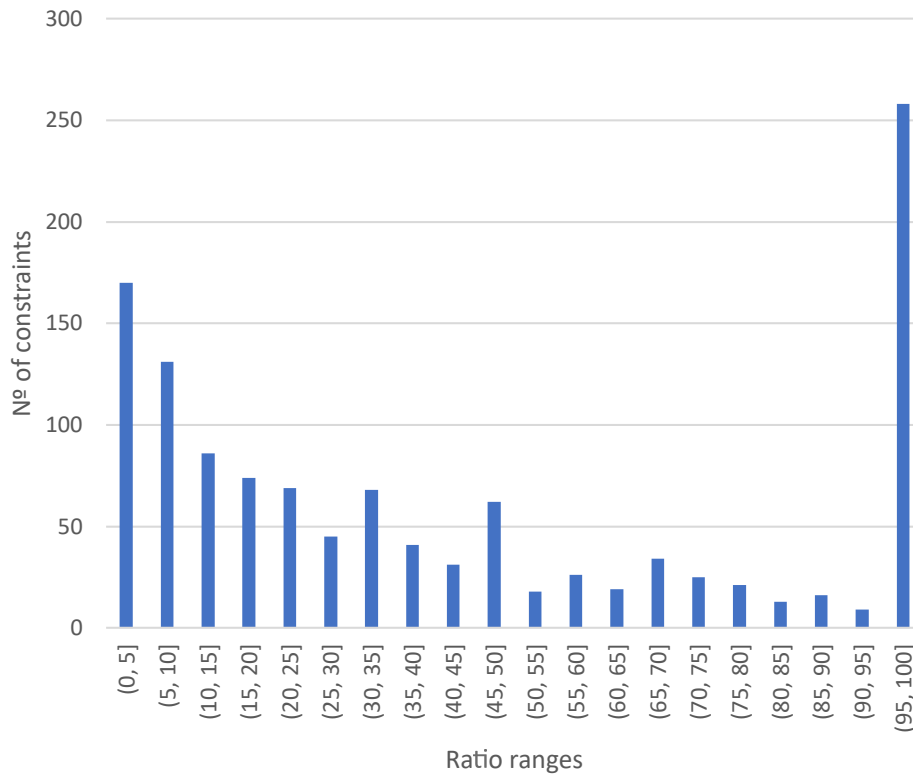
The cardinalities ‘*’ or ‘?’ are never seen in comments. However, note that this is because sheXer does not cast votes for candidate constraints including a zero-case (cf. section 4.2.4).

Note also that the most specific cardinalities (exactly one or an exact range) are more frequent than the less specific ‘+’ cardinality. This makes sense in this context due to sheXer’s configuration. sheXer is aiming to generate TCs as specific as possible. When a shape contains a TC with cardinality ‘+’, it means that there were several candidate TCs with the same node constraint and different exact cardinalities that were transformed into comments. If there had been just one candidate with exact cardinality, then that candidate would had been selected as part of the actual shape’s TCs. Thus, in any execution of sheXer using a similar configuration, it will be observed that the number of comments containing exact cardinalities is, at least, twice the number of comments with ‘+’.

A similar logic can be applied to explain the distribution of node constraints among comments. When there are several candidate TCs with a different shape label as node constraint, the macro **IRI** is selected as the node constraint of the TC. However, in this case, it cannot be generalized that the number of shape label node constraints among comments is at least twice the size of **IRI** node constraints in any execution of sheXer. Whenever sheXer analyzes an object-to-object triple, if the non-focus node is not associated to any shape, a candidate TC with the macro **IRI** gets a positive vote, but no positive votes are generated for candidate TCs referencing a shape label.

In Figure 5.10, we show the distribution of scores associated to each TC (each score indicates the ratio of instances conforming with the TC). Note that any TC including a zero-case cardinality conforms with every instance. In that case, the score accounted for a TC is not 100.0, but the score that this TC would have had if we exclude the zero-case. Such score can be always found in the first text comment

FIGURE 5.10: Distribution of trustworthy scores among TCs in SEITMA-L results.



associated to the TC. In Figure 5.11, we show a similar score distribution for comments.

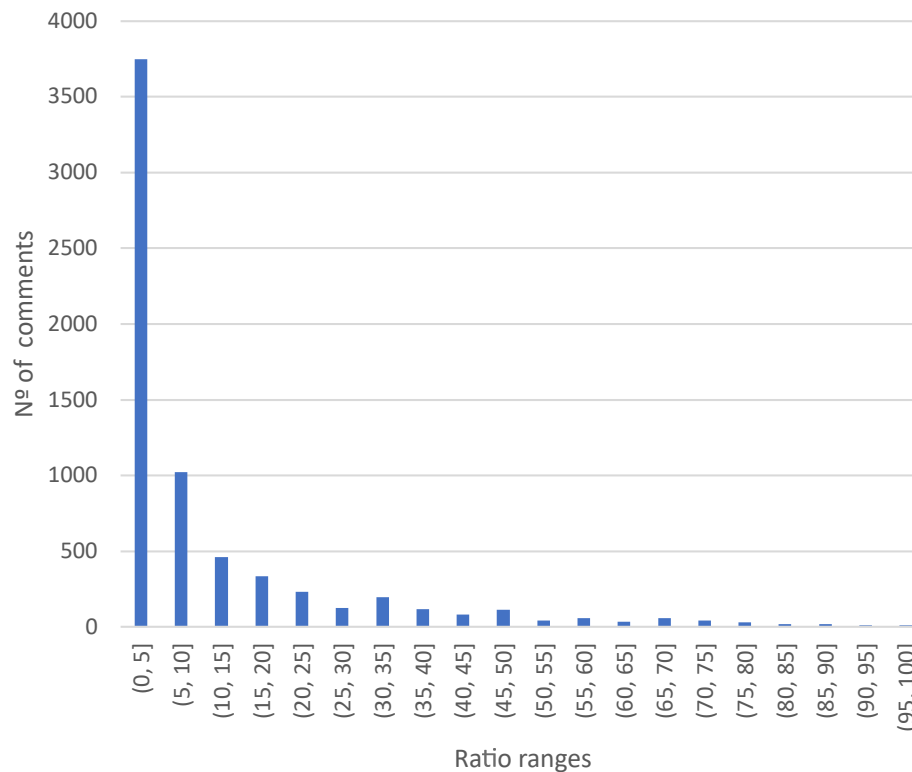
One can see two main differences between Figure 5.10 and Figure 5.11:

- The most frequent score range among TCs is (95, 100]. In contrast, this range is the most infrequent one among comments. Two facts explain this observation. On the one hand, every typing is indicated with a TC of score 100.0 with no associated comments²⁵. Those TCs represent 200 of the total 258 cases in which the score 100.0 is used. On the other hand, several TCs with a score of 100.0 generated for shapes with few target nodes express the most precise cardinality and node constraint possible (a shape label with an exact cardinality). Those TC do not have associated comments either.
- In general, the actual TC's score is higher than the comments' score. This observation can be extrapolated to any sheXer execution, as the score of each TC is always the maximum score among its associated candidate TCs.

Except for the (95, 100] range among the TCs, both figures show a similar trend: they describe a curve in which the lower ranges are more frequent than the greater ones. The curve among comments' scores is more pronounced: the number of comments with a score in (0, 5] is higher than the sum of comments with scores in any other range.

²⁵We are computing the comments that represent a whole TC discarded. A non-discarded TC with a 100.0 score does have an associated comment indicating this score. However, this later type of comment was not accounted in the statistics generated in Table 5.2 nor considered in the rest of figures, as it lacks its own node constraint and cardinality definitions.

FIGURE 5.11: Distribution of trustworthy scores among comments in SEITMA-L results.



Such distributions of scores, especially the one observed among TCs, indicate that the RDF data is heterogeneous and it is more frequent to find patterns that are observed among a reduced number of target instances. This can happen because the input target data of SEITMA-L is indeed heterogeneous or because the triple generator module fails to catch part of the knowledge. Although it is hard to quantify which is the proportion in which these two causes affect the data, examples of both have already been mentioned among the TCs of the *:Band* shape shown in Figure 5.6.

5.4.1.3 Discussion about SEITMA-L features

SEITMA-L is able to extract promising shapes based in Wikipedia abstracts. An obvious downside of our current approach is that it is tightly connected to the Wikimedia ecosystem. However, we think that it could be possible to adapt SEITMA-L to sources different to Wikipedia, as long as those data sources have the following features:

- It must be possible to perform NER and EL so the entities in the text are not only recognized, but also associated with an URI and typed using classes of a controlled vocabulary.
- The set of candidate relations (properties) to extract between the named entities of the target text should be known a priori. Also, those relations must define domain and range constraints which allow to filter candidate mentions w.r.t. to the types of the entities detected during the NER and EL stages.

- The text corpus to analyze should be structured in small units that can be identified with the description of a certain entity/topic. This entity should be typed with classes of the vocabulary used during the EL stage.
- The writing style of the target text pieces should be reasonably homogeneous. The text samples are expected to be focused on the topic they describe and to contain similar types of notions expressed about different examples of a certain class or topic.

Even if it is possible to expect such features in some examples of technical, scientific, or encyclopedic-like entries, the use of SEITMA-L with a non-Wikipedia dataset would require a specific study of feasibility.

The shapes extracted from Wikipedia with SEITMA-L have some interesting features:

- The classes and properties generated belong to the *dbo* namespace, which is a well-known and well-defined vocabulary.
- The simplicity of the vocabulary and the nature of the relations that can be mined with the LAREWA-based proposals lead to shapes concise and easy to interpret.
- Even if the triple constraints are in general unspecific w.r.t. cardinality and node constraint, their associated comments allow for a better understanding of the type of nodes that can be expected to find in the focus nodes' neighborhood.

The performance of SEITMA-L relies on the correctness and completeness of DBpedia in several ways:

- **Entity types.** A mention can be considered candidate for a triple as long as the type of the depicted entity matches the domain a range defined for the predicate. This means that, in case an entity is not typed or wrongly typed, it cannot be used as candidate even if the sentence in which it is used contains truthful knowledge.
- **Knowledge representation.** Sometimes, the training data on which the models rely is not representative of the actual knowledge contained in the abstracts, because the relations used in the abstracts are not stated in DBpedia. Properties for which it is not possible to find enough examples in DBpedia cannot produce reliable learning models. The knowledge associated to those properties does not generate triples and, thus, does not have any effect on the final shapes.
- **Domain and range definitions.** The prototype needs domain and range definition for any property involved in the triple extraction stage. Properties lacking those elements are discarded, and triples that define too vague domain and range generate too many candidates.
- **Constraint violations.** Some triples in DBpedia cause constraint violations w.r.t. to the domain/range definition of the triple's predicate. This causes the exclusion of actual DBpedia triples in the RDF content generated by SEITMA-L's triple extractor.

Issues occurring due to those DBpedia features can have negative effect on the results. Most of those issues are related with the distant supervision approach used to gather training data. As already stated, just 54 properties generated at least one model to make triple predictions, and some of those properties have been detected to be involved in wrongly predicted relations.

A way to tackle most of the issues mentioned would be to feed SEITMA-L with training data annotated by humans. We think that this kind of training data would be a more trustworthy source of knowledge. This approach would allow for using positive examples even if they represent pieces of knowledge which are not in DBpedia. Besides, it would avoid false negatives caused because a certain piece of knowledge is not in DBpedia. Also, it could be feasible to combine human annotators and automatic processes to generate negative examples.

We also think that SEITMA-L would be able to generate reliable models for more properties using human annotated training data. During our experiments, we set 13 as the number of minimum examples to train and evaluate each classifier. Despite this low value, we were able to extract enough training data for only 66 classifiers. This could be corrected using enough data labeled by humans, making it possible to, at least, evaluate classifiers for any property as long as 1) its semantics are actually used within the target abstracts, and 2) it provides domain and range definitions that allow to generate candidates for the triple extraction stage.

Note that our current implementation of SEITMA-L's triple extractor is fully focused on the relation between entities (URIs). This means that no triples using literals are produced. This could be fixed using human annotators as follows:

- The human annotators should identify literal values linked to the abstract's entity by means of a relation r .
- An *xsd* type should be assigned to each literal. This type can be used to look for candidates in the triple extraction stage.
- Some additional features related to the actual value of the literal could be used to train the classifiers and to predict triples.

Needless to say, using human-annotated would affect in a negative way the prototype's scalability. With our current system based on distant supervision, a user only needs to choose some target classes and entities, and SEITMA-L performs every operation in order to extract example data, train models, generate triples, and produce shapes. In contrast, the need of human annotation of positive examples would suppose a bottleneck to expand the extraction process to compute new classes or be executed in new environments, such as different Wikipedia chapters.

Note also that the SEITMA-L's approach relies on the Wikipedia community's precision and recall identifying mentioned entities and linking them correctly with its corresponding Wikipedia page. Even if it is assumed that the entity labeling process in Wikipedia has been performed with the best possible quality, there are some issues virtually unavoidable within SEITMA-L basic workflow:

- If there is no Wikipedia page for an entity mentioned in the text, then that entity cannot be linked. Thus, it cannot be recognized to generate a candidate triple.
- According to Wikipedia standards, the community is supposed to use a link between pages just for the first mention of an entity in the text. This means

that, in case an entity is mentioned twice in an abstract, just the first mention will be recognized and possibly used to generate candidate triples.

Labeling errors related to the mentioned situations can have a negative effect in the triple generation stage. However, as long as those errors are not generalized among the target abstracts, we think it would have little effect on the obtained shapes. They can decrease the score of some TCs or comments, but they would not cause a certain TC to be removed from a shape as long as the number of target entities used to extract that shape is large enough.

5.4.2 Experiments with SEITMA-F

This section is structured as follows: first, we explain the inputs and settings used. Then, we analyze the outputs using an example shape and statistical analyses. Finally, we discuss the potential of SEITMA-F.

5.4.2.1 Inputs and settings

SEITMA-F requires two types of inputs:

- **Target abstracts.** We have extracted shapes from the set of abstracts described at the beginning of section 5.4, i.e., the abstracts of the 300 most important instances from the 200 most important classes in DBpedia.
- **DBpedia typings.** We have used the same typings described earlier in section 5.4, which are the ones corresponding to the target instances within the *dbo* namespace.

SEITMA-F also requires some extra settings to configure sheXer and the FRED Consumer sub-module:

- **Gap between FRED requests.** This gap ensures that the maximum rate of requests per minute to the FRED API is never exceeded. We set this value to 15 seconds.
- **Maximum number of requests per day.** It ensures that the maximum requests per day performed to FRED API is never exceeded. We set this value to 7500 petitions.
- θ_s , **sheXer's acceptance threshold** (cf. section 4.4.6). We set $\theta_s = 0$, which is the same configuration used in SEITMA-L's experiments.

The number of temporal triples and final shapes generated with SEITMA-F is much bigger than the data volume generated with SEITMA-L. The number of classes in the RDF content increased because FRED generated custom typings for many entities and sheXer works in *all_classes_mode* (cf. section 4.4).

We think that using a single file to contain all the shapes could be confusing due to the large number of elements generated. For this reason, instead of executing SEITMA-F to extract every shape from every target instance in a single iteration, we have split the experiments and performed one iteration per class. Each iteration extracts shapes using the target instance of a certain class.

5.4.2.2 Outputs

This subsection is organized in two parts. First, we show and comment an example shape produced by SEITMA-F which let us highlight some of our prototype’s features using a real example. Then, we summarize the overall results with a statistical analysis of the shapes produced by SEITMA-F.

Example shape - In Figure 5.12 we show a shortened version of the shape `:Band` produced by SEITMA-F when executing it over the target instances of `dbo:Band`. The content removed from the original shape `:Band` has been marked with a “# ...” comment. The version of `:Band` shown in Figure 5.12 is much smaller than the original shape, which is 686 lines long.

Note that the shape `:Band` produced by SEITMA-L is written in 109 lines, and note also that SEITMA-L’s experiments were designed to extract every shape in a single iteration, which increases the chances of finding extra inverse TCs and extra comments with specific node constraints. This fact raises a clue about how different can be the amount of data generated by SEITMA-F and SEITMA-L.

Note also that SEITMA-F only keeps the triples generated by FRED in case this subsystem is able to successfully identify a certain node with the correspondent target DBpedia URI. Then, even if the system receives 300 target abstracts, it may generate triples for only a reduced slice of them. In the case of `:Band`, FRED was able to successfully match 50 individuals with its corresponding DBpedia URI. Then, the shape `:Band` was extracted by using knowledge of just those 50 entities.

Even with this, the number of triples that FRED is able to extract from a certain abstract is, in general terms, higher than the triples extracted with LAREWA²⁶. Using only those 50 target abstracts, SEITMA-F is able to generate a total of 182 TCs for the shape `:Band`.

The sample population used is heterogeneous, so there are just two TCs conforming with every node, i.e., having a score of 100.0. Most of the TCs use ‘*’ and ‘?’ cardinalities that include a zero-case.

As one can see, `:Band` has several typing TCs. Every shape extracted with instances of a target class includes a typing TC with score 100.0 referring that target class. However, many other typing TCs use classes which are not in the `dbo` namespace. Such classes are mainly defined in the `fred` and `schema` namespaces (see lines 8, 10, 37, and 39 in Figure 5.12). The triples causing those types to appear in the shape are generated by FRED and kept by SEITMA-F because they can be valuable knowledge. For example, the TC in line 39 let us know that 8% of the `:Band` instances (at least) are English rock bands. Those types can include also noisy or wrong knowledge though. For example, 8% of the target entities are also detected to be instances of `fred:Song`, which seems a wrong association.

We can observe some properties with quite general semantics at the top of `:Band`’s TCs, such as `framerster:playsRoleIn`, `dul:associatedWith`, and `boxer:agent`. This is something that can be observed in most of the shapes generated by SEITMA-F. Those TCs usually include the node constraint `IRI`. Such general semantics makes them a poor tool to describe or validate a shape, as most of the target instances conform with them. However, the comments associated to those TCs contain valuable knowledge. For example, one can know that bands are associated to albums and songs with the comments in lines 15 and 16. Also, that they can be agents of releases, as indicated in line 26.

²⁶This affirmation will be developed later in this section.

FIGURE 5.12: `:Band`. A shortened example of a shape produced by SEITMA-F

```

1
2 :Band
3 {
4   framester:playsRoleIn IRI +; # 100.0 %
5     # 20.0 % obj: IRI. Cardinality: {2}
6     # ...
7   rdf:type [dbo:Band] ; # 100.0
8     %
9   rdf:type [schema:Organization] ?;
10     # 86.0 % obj: schema:Organization. Cardinality: {1}
11   rdf:type [schema:MusicGroup] ?;
12     # 86.0 % obj: schema:MusicGroup. Cardinality: {1}
13   ^ dul:associatedWith IRI *;
14     # 80.0 % obj: IRI. Cardinality: +
15     # ...
16     # 34.0 % obj: @:Song. Cardinality: +
17     # 34.0 % obj: @:Album. Cardinality: +
18     # 32.0 % obj: @:SecondAlbum. Cardinality: +
19     # ...
20   dul:hasQuality IRI *;
21     # 76.0 % obj: IRI. Cardinality: +
22     # ...
23   ^ boxer:agent IRI *;
24     # 68.0 % obj: IRI. Cardinality: +
25     # ...
26     # 34.0 % obj: @:Thing. Cardinality: +
27     # 26.0 % obj: @:Release. Cardinality: +
28     # ...
29   ^ fe:Place.coming_to_be IRI *;
30     # ...
31   ^ fe:Seller.commerce_sell @:Commerce_sell *;
32     # ...
33   dul:hasDataValue xsd:nonNegativeInteger ?;
34     # ...
35   ^ fe:Cook.cooking_creation @:Cooking_creation *;
36     # ...
37   rdf:type [fred:Song] ?;
38     # 8.0 % obj: fred:Song. Cardinality: {1}
39   rdf:type [fred:English_rock_musicBand] ?;
40     # 8.0 % obj: fred:English_rock_musicBand. Cardinality: {1}
41     # ...
42   boxing:declaration IRI *;
43     # ...
44     # 2.0 % obj: @:InternalStruggle. Cardinality: {1}
45     # ...
46   fred:secondAlbumOf IRI *;
47     # ...
48   ^ fred:musicOf @:Music *;
49     # ...
50     # ...
51   ^ fred:albumOf IRI ?;
52     # ...
53     # ...
54   ^ fred:commercialFailureOf IRI ?;
55     # ...
56     # ...
57 }
58

```

The number of different node constraints in comments for such general properties can be quite big when using enough target entities though. For example, the number of different node constraints among comments associated to the TC `(^ dul:associatedWith IRI *)` in the non-shortened version of the `:Band` shape is 40.

We can observe some properties with more concise semantics too. Some of them are related the music domain, such as `fred:albumOf`, `fred:commercialFailureOf`, or `fred:musicOf`, while some others, such as `fe:Cook.cooking_creation`, are not. The first group of TCs helps to determine the frequency of relations that one may expect to find in any instance of `Band`. The later group reveals knowledge which could be hardly expected to be found a priori, even by domain experts.

Note that the semantics for some of those properties are sometimes subsumed by the semantics of another property. For example, the properties `fred:albumOf` and `fred:secondAlbumOf` are found among `:Band`'s TCs. One can intuitively understand that any *second album* of a certain band is also an *album* of this band. The notion of second album should probably be represented using the property `fred:albumOf` and some kind of reified schema. Performing such a knowledge refactoring in an automatic heuristic-based manner is feasible. Indeed, FRED already performs such a task for some properties detected to be compound (cf. section 5.3.2.1). However, the current SEITMA-F implementation is not able to decompose every compound property.

Some other observations that seem to correlate with the specificity of the property's semantics are the TC's score and its number of comments. The more general a certain property is, the more comments and high scores have (except for typing TCs, which do not have associated comments with extra node constraints).

This cannot be clearly observed in the version of the `:Band` shape shown in Figure 5.12 due to the content removed. Let us refer to some representative examples of already mentioned properties of this shape though. The TCs with the properties `dul:associatedWith` (score 80.0) and `boxer:Agent` (score 68.0) have 47 and 52 associated comments respectively, and they both use `IRI` as node constraint. In contrast, properties with less general semantics, such as `fe:Cook.cooking_creation` and `fred:musicOf` have 3 associated comments each and use a node constraint that refers to a shape label. There seems to be a reasonable inverse relation between the specificity of a certain TC and the number of observed examples for such relation. As already explained in the analysis of SEITMA-L's experiments, fewer examples usually imply to be able to extract more precise TCs, which have lower score and less associated comments.

We can observe that the TCs using properties with general semantics usually have a better score than TCs using properties with concise semantics. This happens due to FRED's schema to represent n-ary relations. Frequently, the properties with broad semantics are used to link the focus node with an element which is related to some other pieces of knowledge that, all together, can represent the whole meaning of a certain statement in a frame. The current SEITMA-F implementation scratches that meaning by detecting the type of the main node in such n-ary relations, which is useful to produce informative comments associated to the TCs. The rest of the elements in the frame are discarded by SEITMA-F, in an attempt to generalize structures that could apply to any band instance rather than a specific statement about a given band.

The properties which are fundamental for such general schemata are frequently used in the graph. Therefore, TCs using those properties usually have a high score

	Target shapes	Every shape
Number of shapes	136	15141
Number of TCs	9974	123287
Number of comments	27717	255799
Avg. constraints per shape	73.34	8.14
St.dev. constraints per shape	78.24	7.56
Top n° of constraints in a shape	709	709
Avg. comments per shape	203.80	16.89
Std.dev. comments per shape	242.42	19.65
Avg. comments per constraint	2.78	2.07
Std.dev. comments per constraint	2.65	2.33
Avg. ratio constraints	9.00	70.12
Avg. ratio comments	5.19	54.69

TABLE 5.3: Statistics about shapes in SEITMA-F results

but broad cardinalities and node constraints (typically `*` and `IRI`). In contrast, properties with more precise semantics are found among less target instances, causing them to have lower scores.

A difference observed between the `:Band` shape produced by SEITMA-F and the one produced by SEITMA-L is that the SEITMA-F's `:Band` seems to have a balanced distribution of direct and inverse TCs, or even more inverse TCs if we ignore typings. This is caused by the FRED's data schemata already mentioned. Focus nodes are frequently involved in subgraph structures designed to represent the meaning of a certain statement with reified schemata. In such schemata, the focus node is used as the object of a triple whose subject is connected to some other pieces of information. Those types of structures make inverse paths from the focus node more frequent in SEITMA-F graphs compared to SEITMA-L graphs.

Shapes' stats - In Table 5.3, we show general data about the shapes obtained during the described experiment with SEITMA-F. The total number of shapes generated with the abstract inputs is 15,141. From those elements, just 136 (0.9%) are related to the actual target classes of the experiment. The rest of elements are consequence of typing triples introduced by FRED. Those shapes are helpful to understand the structure of knowledge related to the main ones.

As one can see in Table 5.3, the average number of constraints and comments per shape is much higher among the target shapes than among the total shapes. We will first provide an independent analysis of the target shapes. Then, we will perform a similar analysis on the total shapes produced by SEITMA-F.

Every shape and RDF content generated during this experiment have been published²⁷

Target shapes' stats - The number of average TCs associated to a target shape in SEITMA-F is more than 12 times the SEITMA-L's average. Also, the average number of comments associated to a target shape in SEITMA-F is more than 6 times the average observed in SEITMA-L. Those differences are mainly caused by two facts. First, the graph size. The number of triples generated by SEITMA-L is 24,376, while

²⁷The shapes can be downloaded at http://data.weso.es/seitma/seitma_f_shapes.zip. The triples can be downloaded at http://data.weso.es/seitma/seitma_f_triples.zip Accessed in 2022/05/03.

SEITMA-F was able to generate 89,971 triples using the same input abstracts. Second, the number of different properties. We were able to train SEITMA-L to generate triples using only 54 different properties. In contrast, the amount of properties seen in SEITMA-F's generated graph is 2,723.

The larger number of triples and property variety of SEITMA-F's content allow for finding richer shapes whose TCs use to have many associated comments covering features observed in few cases.

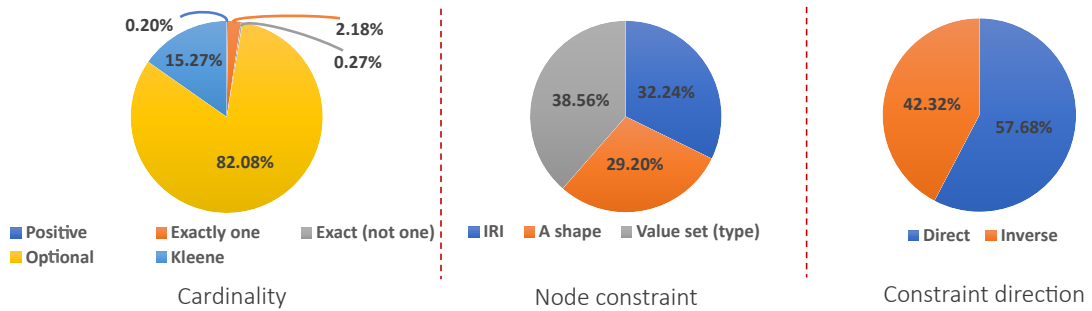
Note that the standard deviations of the mentioned averages are even higher than the average itself. This remarkable dispersion among the number of elements associated to each shape is caused by the different amount of knowledge extracted for each class. Even if SEITMA-F improves the absolute numbers of SEITMA-L w.r.t. number of triples and property variety, the amount of triples extracted for each class is not homogeneous. Several causes lead to that situation:

- Some target class lack enough instances, or the abstracts associated to those instances are too short or cannot be processed.
- A certain sentence parsed as an independent element with FRED needs context to be properly interpreted. Even if FRED performs CRR tasks, sometimes the sentences sent to the tool do not contain enough context to identify that a certain noun or pronoun is used to refer to the abstract's title entity.
- FRED is able to identify the abstract's title entity in a sentence and to extract knowledge about it, but it fails to successfully link such entity with its DBpedia URI via an *owl:sameAs* axiom. In those cases, SEITMA-F discards the sentence's RDF graph. FRED's EL performance seems to decrease with entities representing actual people, so this is a common situation when mining subclasses of *dbo:Person*. For example, classes such as *dbo:Politician* or *dbo:SoccerPlayer* produce empty shapes even if FRED can extract many triples from their instance's abstracts.

We can see in Table 5.3 that average score ratios for TCs and comments generated by SEITMA-F are lower compared with SEITMA-L. However, this average is not caused by a lack of high-scored TCs, but by a large number of low-scored ones that appear due to features rarely observed among the target instances. For example, the shape *:Settlement* generated by SEITMA-F, which is the shape with the highest number of TCs (709), has 28 TCs which are above the average score. This amount of TCs is higher than the total number of TCs of the shape *:Plant*, which is the shape that has more associated TCs among the ones generated by SEITMA-L (18). SEITMA-F's *:Settlement*, in addition to those 28 TCs above the average, has another 682 TCs with a score under 9.00. Although *:Settlement* is an extreme case, it is a frequent situation to have an unbalanced number of high-scored and low-scored TCs and associated comments, which decreases the averages shown in Table 5.3.

In figure 5.13, we show the distribution of cardinalities, node constraints and directions observed among the TCs generated by SEITMA-F. As one can see, cardinalities including a zero-case are predominant. Only 2,18 % of the TCs use a cardinality of exactly one. Most of those cases correspond to typings indicating the DBpedia class used to locate the instances of a target class. The rest of TCs with no zero-case cardinalities are marginal. In general, those TCs use frequent properties of broad semantics or are typings of classes in the *schema* namespace.

FIGURE 5.13: Cardinality, node constraint, and constraint direction w.r.t. focus node of TCs in SEITMA-F target shapes.



The optional cardinality is especially frequent. The reason for finding more ‘?’ cardinalities than ‘*’ is the specificity of many properties generated by FRED. It is frequent that a certain relation r is observed for few instances which generate cardinalities that must include a zero-case. Also, those few instances usually use r a single time. The optional cardinality is the most specific w.r.t. maximum and minimum occurrences to express such situation.

The great amount of typing TCs observed among the target shapes is caused by the classes generated by FRED too. Each target shape is not just related with its DBpedia type, but usually also to many other classes in the *schema* and *fred* namespaces.

The variety of properties produced by FRED causes that there is a balanced number of TCs using IRI and a certain shape label as node constraint. In line with the trends observed for the shape *:Band*, the macro IRI is frequently observed in high-scored TCs. In opposition, low-ranked ones, which are usually generated over a single or a few observations among the target instances, tend to use a precise shape label as node constraint.

Also in line with the trends observed for the *:Band* shape, the direct and inverse observations are quite balanced. As already explained, the increase of inverse TCs in SEITMA-F compared to SEITMA-L is caused by FRED’s patterns to express n-ary relations.

In Figure 5.14 we show cardinality and node constraint distributions among generated comments. Comparing those distributions with the TCs distributions, one can see a general increase of specificity. The predominant cardinality is ‘1’. Regarding node constraints, shape labels are used more than twice more compared to the macro IRI. This trend is mainly caused by the large number of classes generated by FRED. High-scored TCs tend to use IRI node constraints and ‘*’ cardinality. However, those TCs are frequently extracted using a large number of target entities showing many particular cases, which motivates the appearance of comments with precise shape labels and cardinality ‘1’.

In Figure 5.15, we show the score distribution of TCs among the shapes associated to target classes. In Figure 5.16, we show the same distribution among the comments associated to those classes.

Please, note that these two figures have a chart break in the y-axis. In both cases, the number of observations in the range (0, 5] is an order of magnitude higher than the observations associated to any other range. Again, this is caused by the amount of particular cases generated by FRED. The chart break in those figures has been introduced in order to be able to properly compare the rest of the ranges.

FIGURE 5.14: Cardinality and node constraint of comments in SEITMA-F target shapes.

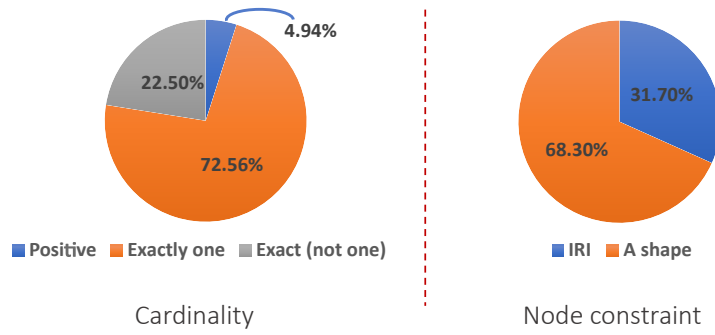


FIGURE 5.15: Distribution of trustworthy scores among TCs in SEITMA-F target shapes.

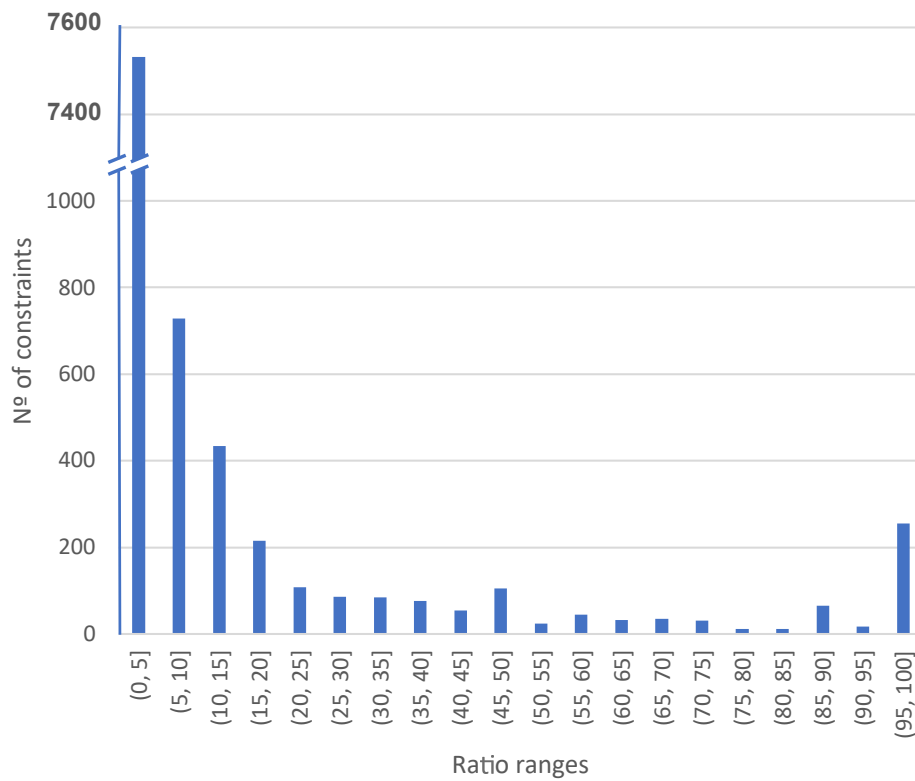
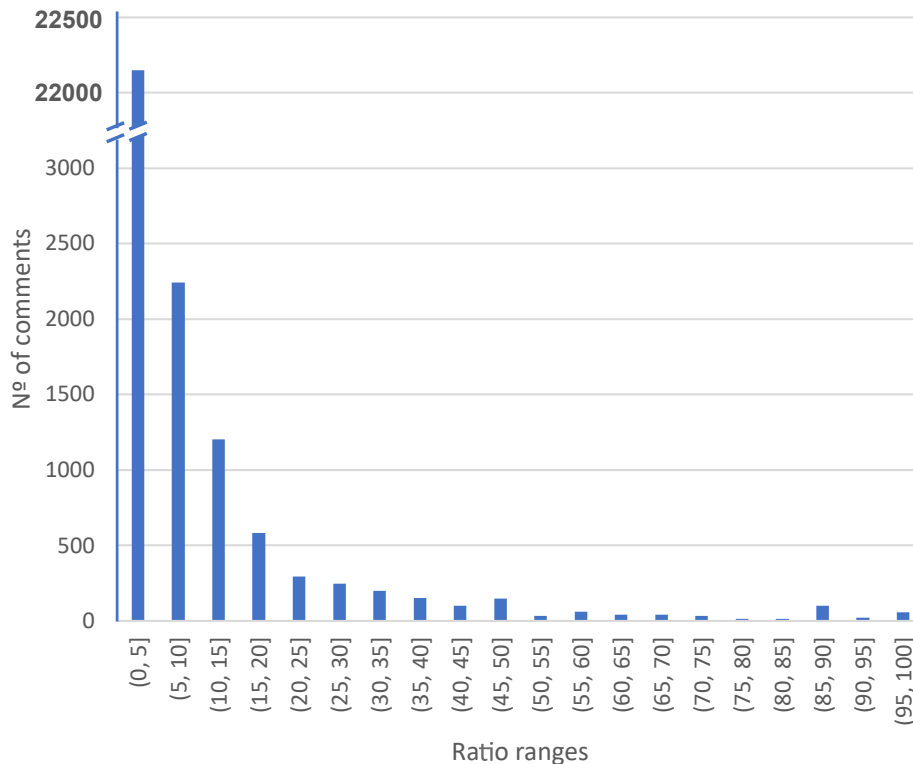


FIGURE 5.16: Distribution of trustworthy scores among comments in SEITMA-F target shapes.



As one can see, both charts tend to have more observations in low scores. This general trend is less clear among the TCs scores though. There is a noticeable proportion of cases accumulated in the range (95, 100]. Such scores appear mainly in three scenarios: 1) TCs with DBpedia typings, 2) some TCs with general properties such as *framerster:playsRoleIn*, and 3) TCs in shapes built with few examples.

Total shapes' stats - In this subsection, we analyze the statistics of the total 15,141 shapes produced by SEITMA-F. In Figure 5.17, we show the cardinality, node constraint, and triple direction observed among the TCs. In Figure 5.18, we show the cardinality and node constraint distribution observed among comments. In Figure 5.19 we offer the TC's score distribution. Finally, in Figure 5.20, we show the score distribution among comments. In this subsection, we will focus on the main differences observed between these figures and the measurements associated to the shapes associated to the target classes.

We will denote the set that contains every shape with S . The shapes of S can be roughly classified in three groups.

- The shapes associated to the target classes studied in the previous subsection. We denote this group with S_{TAR} .
- Shapes that are extracted for some other classes but have still been mined using many instances. Frequently, the instances used for those shapes are also instances of the target classes. They are linked to classes in the *schema* and, sometimes, *fred* namespaces. We denote this group as S_{ALT} . The statistical composition of S_{ALT} and S_{TAR} is similar.

FIGURE 5.17: Cardinality, node constraint, and constraint direction w.r.t. focus node of TCs in shapes produced by SEITMA-F.

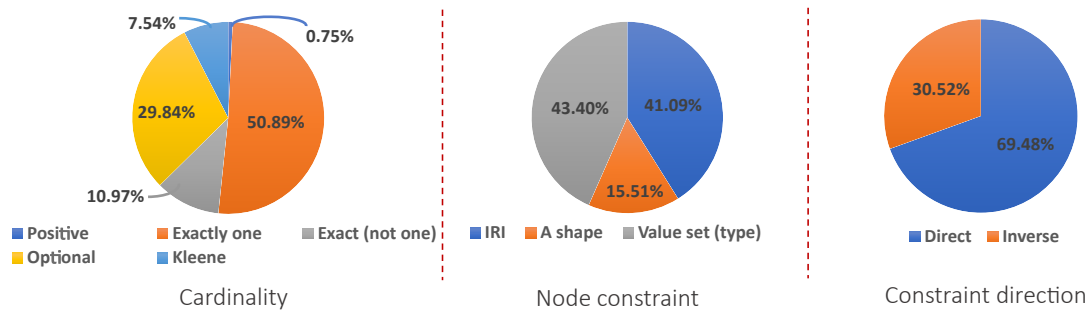


FIGURE 5.18: Cardinality and node constraint of comments in shapes produced by SEITMA-F.

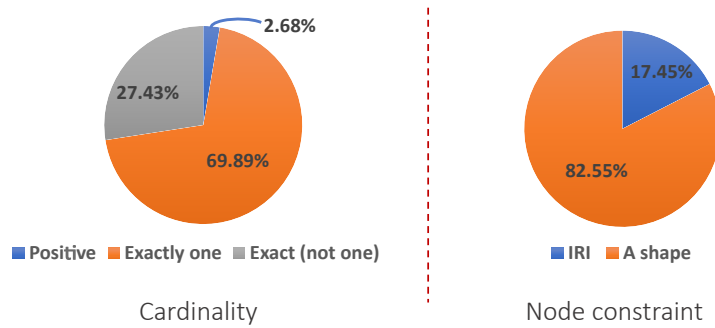


FIGURE 5.19: Distribution of trustworthy scores among TCs in shapes produced by SEITMA-F.

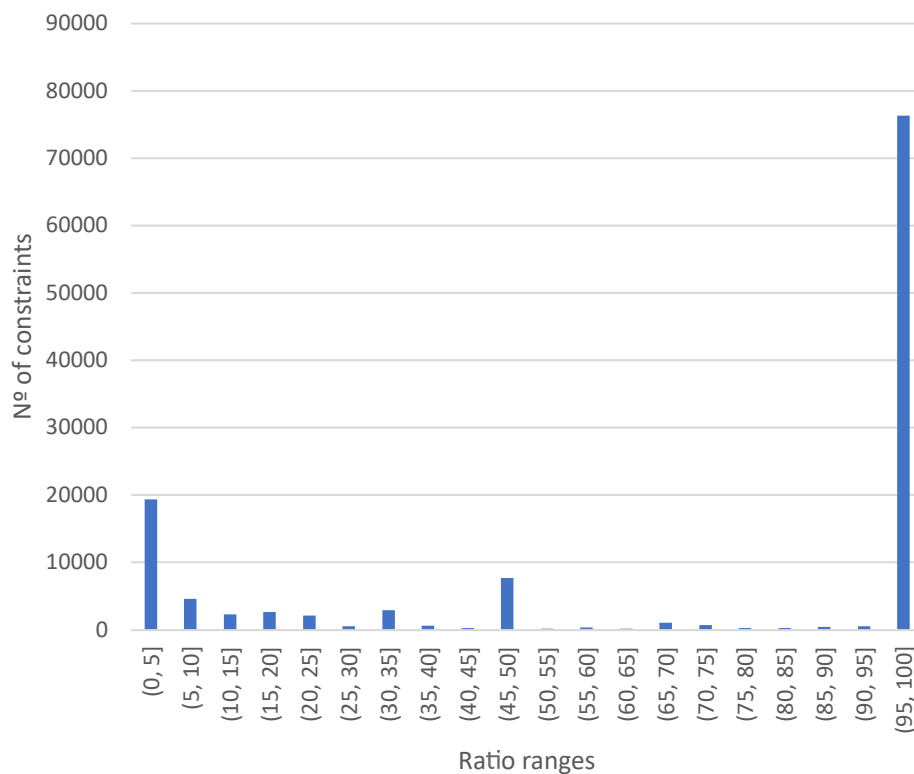
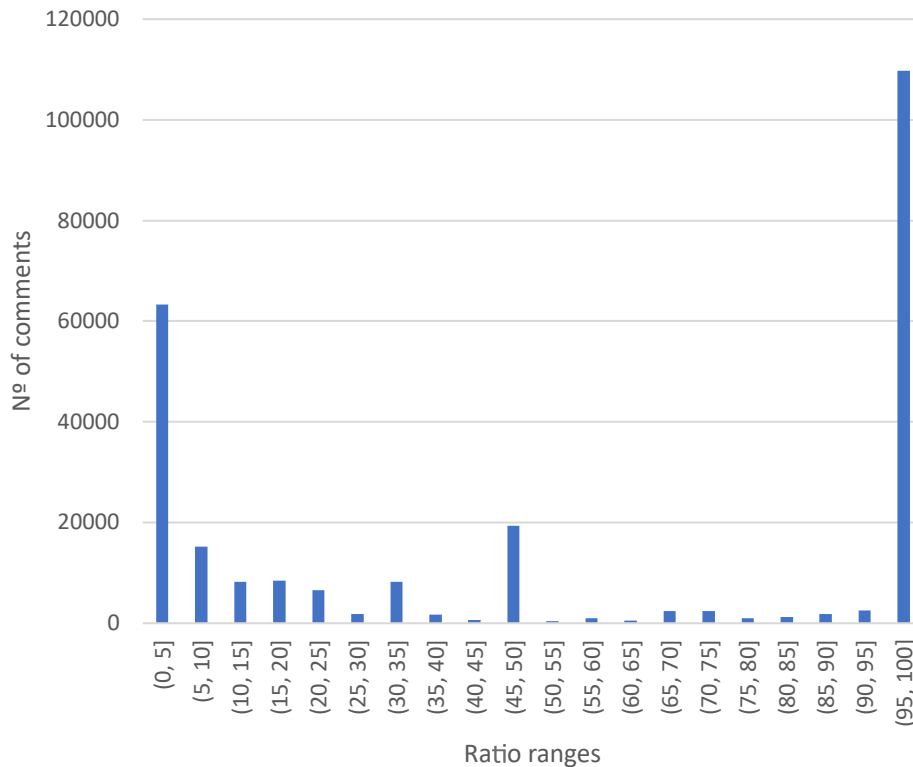


FIGURE 5.20: Distribution of trustworthy scores among comments in shapes produced by SEITMA-F.



- Shapes that are extracted using a few instances linked to some class in the *fred* namespace. We denote this group as S_{MIN} . The features observed among those shapes are different to S_{ALT} and S_{TAR} . Shapes in S_{MIN} tend to have less TCs and comments and higher specificity w.r.t. cardinality and node constraint. Most of the shapes in S belong to S_{MIN} .

We have not defined a strict formal difference that allow to classify a certain shape s as a member of S_{MIN} or S_{ALT} . Such classification of shapes is still useful, as it allows us to avoid verbosity when referring to some group of shapes exhibiting some distinctive features. Most of the statistical differences observed between the figures associated to S and the figures associated to S_{TAR} are caused by the S_{MIN} shapes.

One of the most remarkable differences between S_{TAR} and S is the dominance of cardinalities without zero-case among the TCs of S . Just the cardinality '1' is found in more than half of the TCs. Cardinalities without zero-case are unusual in shapes extracted using many instances such as S_{TAR} and S_{ALT} . However, most of the shapes produced in our experiments belong to S_{MIN} . This increases the chances of observing features that actually conform with every individual extracted to generate a shape without needing to use a zero-case cardinality.

The proportion of typing TCs tend to be higher in S_{MIN} . This has an effect on measurements related to node constraint and direction distributions of S . The proportion of value sets (typings) is increased. Also, the proportion of direct TCs is increased too, as all typings are direct.

S_{MIN} shapes also have a decisive effect on the comment distributions within S . The lack of a large number of different observations for a given feature causes that there are less comments using the macro `IRI` with different cardinalities. Instead,

the comments contain a higher proportion of specific shape labels and ‘1’ cardinality. This explains why the use of the macro `IRI` is less frequent among comments when it is actually more frequent among TCs.

The last big difference between the statistics of S and S_{TAR} can be observed in the score distributions. In the figures 5.15 and 5.16 related to S_{TAR} , we observe that the charts are dominated by the cases occurring in the $(0, 5]$ range. In contrast, in figures 5.19 and 5.20 related to S , we can see that the majority of cases are concentrated at the $(95, 100]$ range. This is caused by the amount of TCs with ‘1’ cardinality, as they are always associated with a 100.0 score.

Excluding the $(95, 100]$ range, the trend observed in these two charts is still a curve that concentrate cases in low ranges. However, both charts have a peak of observations in the $(45, 50]$ range. This is explained by the large number of shapes in S_{MIN} that have exactly two instances. With this number of instances, the possible scores for any TC or comment can be either 50.0 or 100.0. In case a certain feature conforms with just one of the instances, then the score is 50.0.

Note that a similar fact can be observed in the range $(30, 35]$ in both charts. Here, this is motivated by shapes extracted using exactly 3 instances, in which many features are observed for just one of the instances. Those cases produce scores of 33.33.

5.4.2.3 Discussion about SEITMA-F features

In this experiment, we have used SEITMA-F to extract shapes from Wikipedia abstracts. However, unlike SEITMA-L, SEITMA-F could be easily executed with different data sources. FRED, which runs the core processes within the triple extraction submodule, relies on techniques that are independent of our experiment’s context.

Our current SEITMA-F prototype would require the following minimal adaptations in order be able to extract shapes from a different text source:

- SEITMA-F uses an abstract normalizer submodule which is specifically designed to adapt Wikipedia content written in markdown syntax. This module should be adapted to features of the new target corpus.
- Currently, the *Graph filterer* submodule discards any subgraph generated by FRED if it does not contain the DBpedia URI of the target abstract’s entity. This module should be adapted to perform a different filtering strategy.
- The *Type injector* submodule adds triples related to the Wikipedia abstracts’ entities. Again, the typing triples injected to the graph at this stage should be adapted to the target application scenario.

In comparison to SEITMA-L, the number of shapes, TCs, and comments extracted with SEITMA-F can be even overwhelming. As it has been shown, the shapes extracted with SEITMA-F can reach a high level of detail and specificity. Nevertheless, using SEITMA-F on a real scenario would probably require a different configuration of sheXer’s θ_s . $\theta_s = 0$ allow SEITMA-F to extract every feature observed in any target instance, which causes the final shapes to include particular cases observed among few seed entities. This did not seem an issue with SEITMA-L shapes, as that prototype produced a much lower number of constraints per shape and, in every case, it used a controlled vocabulary. In contrast, the triple extraction performed with FRED in SEITMA-F can be too precise. Many particular cases were included in the shapes obtained during our experiments, and such particular cases may not be representative of the ideas that one may expect to find in a shape related to a general abstract concept (class). On the contrary, they can cause relevant

FIGURE 5.21: Section of the shape `:Artist` produced by SEITMA-F

```

1
2 :Artist
3 {
4 # ...
5 rdf:type [dbo:Artist] ; # 100.0 %
6 ~ boxer:agent IRI *;
7 # ...
8 # 12.5 % obj: @:Employ. Cardinality: {1}
9 # 12.5 % obj: @:Continue. Cardinality: {1}
10 # 12.5 % obj: @:Despite. Cardinality: {1}
11 # 12.5 % obj: @:Seem. Cardinality: {1}
12 # 12.5 % obj: @:Reluctant. Cardinality: {1}
13 # 12.5 % obj: @:Discuss. Cardinality: {1}
14 # 12.5 % obj: @:Wear. Cardinality: {2}
15 # 12.5 % obj: @:Perform. Cardinality: {1}
16 # 12.5 % obj: @:Attract. Cardinality: {1}
17 # 12.5 % obj: @:Cultivate. Cardinality: {1}
18 # 12.5 % obj: @:Launch. Cardinality: {1}
19 # ...
20 }
21

```

FIGURE 5.22: Partial example of FRED's output.

```

1
2 :bob a dbo:Artist .
3
4 _:1 boxer:agent :bob ;
5     boxer:patient :an_album ;
6     #... Some other knowledge could be linked to _:1
7     a fred:Launch .
8
9 :an_album a schema:Album .
10

```

shape features to go unnoticed among the many TCs mined from particular cases. Nevertheless, determining an adequate value for θ_s is necessarily linked to the user intentions on real scenarios.

The strategies used during the execution of SEITMA-F's *Triple filterer* are quite straight-forward. More sophisticated graph adaptation strategies could be performed. Let us focus on some very frequent properties in the resulting shapes, such as `boxer:agent` or `boxer:patient`. With our current implementation, the valuable knowledge of TCs using that kind of properties can be found among their associated comments, which provide frequency scores of specific shape labels. However, it is feasible to use mappings for sub-graph structures using such generic properties. In Figure 5.21, we show a shortened version of the shape `:Artist` produced by SEITMA-F. As one can see, most of the shape labels associated to the property `boxer:agent` are verbs that could be mapped into properties. In Figure 5.22, we show some RDF example that could have produced some of the features observed in the shape `:Artist`. In figure 5.23, we show a possible mapping for this structure and describe the steps performed to achieve such mapping.

With such a mapping, some knowledge attached to the original BNode could be lost, but the shape obtained by mining the neighborhood of the node `:bob` would be

FIGURE 5.23: Possible mapping for the content shown in Figure 5.22.

```

1
2 # We remove the BNode _:1.
3 # The type of the BNode is transformed into a new property.
4 # The boxer:agent and the boxer:patient are directly linked.
5
6 :bob a dbo:Artist ;
7     :launch :an_album .
8
9 :an_album a schema:Album .
10

```

more specific. It would not contain a TC with the property *boxer:agent*, but several TCs with properties linked to *fred* types. Note also that those new TCs could include comments associated to other specific shapes labels (such as a potential *:Album* shape for the content shown in Figure 5.23). It could be feasible to implement similar mappings using structures associated to elements of known vocabularies already included in FRED’s output, such as VerbNet verbs or FrameNet senses.

A limitation of the current SEITMA-F implementation is its inability to perform CRR tasks whenever the target node of an iteration is not explicitly stated in the parsed content. In order to tackle this issue, it could be possible to implement an extra layer for post-processing FRED outputs. The nature of such layer would be related to the specific context application of SEITMA-F.

A possible heuristic-based approach to improve the CRR of graphs extracted from DBpedia abstracts could consist in mapping nodes referring to the type of the abstract’s entity to its DBpedia URI. For example, let us suppose that SEITMA-L is analyzing the abstract of London city, and it finds a sentence such as “*This city was founded by the Romans*”. After extracting content with FRED, we could assume that those nodes whose type is *fred:City* and do not seem to be linked to an specific city (its label is a neutral name such as *fred:city_1*, they are not equated to external URIs with *owl:sameAs* axioms, etc.), can be interpreted as indirect forms of London. Then, we could merge the content related to those city nodes into the graph containing direct statements about the node *dbo:London*. Similar approaches for CRR applied to distant supervision environments have already been studied [282].

5.5 Discussion on use cases

The main purpose of shape languages is to serve as validation and documentation tools for RDF graphs. The automatic extraction of shapes from RDF sources is gaining attention from researchers during the last years, as these shapes can be useful in several ways for the users/maintainers of RDF sources. On the other hand, most of the times, the benefit of having shapes which describe text pieces instead of RDF content is not direct. One cannot expect an average user of a text-oriented platform to be interested nor perceive potential uses from code on a formal language based on RDF concepts. In this case, the benefit of extracting shapes is indirect, i.e., it lies on applications that could be built on top of such shapes.

In this section, we will introduce several use cases in which SEITMA can be beneficial. We will divide the use cases in two groups. First, we will mention some scenarios related to Wikipedia. Then, we will discuss use cases related to other data sources.

5.5.1 Use cases for Wikipedia

5.5.1.1 Automatic generation of templates

Should a user want to write the abstract of an entity of a given class, and having a shape available for that class, then a first textual draft could be automatically produced based on that shape.

Such a draft could consist of a text template with some gaps or placeholders. The user could replace those gaps with appropriate values. That template would consist of a textual representation of the shape's TCs. Labels and descriptions associated to the TCs' properties could be used for the generation of such a template.

It is also reasonable to think about an application to produce forms instead of text templates with placeholders. Such a form would ask the user to fill in some data values associated to the properties found among the TCs of a certain shape. The node constraints of those TCs could also be used to check whether the values introduced by the user are valid. There are existing approaches to automatically transform shapes into forms, both for SHACL [113] and ShEx²⁸.

For example, one may want to add a new page of a certain band in Wikipedia. If it exists a shape `:Band` with the most usual features of any band instance, an application could generate an automatic text draft/form for an abstract with content suggestions. Those suggestions may include properties such as hometown, active/-former members, record labels, etc. This first abstract may be helpful to save some writing time. Moreover, it may give a clue to the editor of the new band's page about what type of information is usually found in a band's abstract. Those clues can help the editor to not forget including any key piece of information in the new abstract.

It could be also feasible to implement applications which are able to fill in some of the template's placeholders by using knowledge in existing KGs, such as DBpedia or Wikidata. This could be possible as long as:

1. The entity described in this new Wikipedia page has an URI on this external source.
2. The properties used in this external source are the same properties used in the shape which generates the template or, at least, it is possible to map the shape properties to the source's properties.

Note that, with such an approach, the content generated would be based on information from external sources. Nevertheless, the template structure would still be entirely based on Wikipedia abstracts, as the shapes used to generate such templates would have been obtained by mining Wikipedia.

The verbalization of formal languages to create textual content has already been explored by Wikimedia's community with the project Abstract Wikipedia[303]. The aim of this project is to write articles using a work-in-progress formal syntax, so this syntax can be automatically translated into different languages. The success of such a project may help to remove cultural barriers to access Wikipedia knowledge. The same content would be available for any language, as long as there is a translator implemented for such language. Abstract Wikipedia seeks for a higher expressiveness level than shape languages though. The syntax used for this project must be able to capture language hints, so the quality of the translated content is not different to quality of the current Wikipedia. Nevertheless, until such syntax is developed,

²⁸This feature is available in the portal <https://rdfshape.weso.es/shexConvert>. Select "Shape-Forms" within the **Target engine** combo-box to see a demo. Accessed in 2022/05/03.

shapes can generate less expressive pieces of text that can contribute the Abstract Wikipedia general goals.

5.5.1.2 Content suggestion

SEITMA's triple extraction stage could be ran independently over abstracts of class instances which already have an associated shape. In case the triples extracted from those target abstracts do not conform with their class shape, the constraint violations could be used to suggest content additions or corrections. Such suggestions could be implemented in different levels:

- **Bots.** The process to check whether an abstract conform with a class shape could be automatically performed by bots in every instance of a class with a shape. Such bots could produce automatic alerts indicating the constraint violations detected in different ways (comments in the *discussion page*²⁹, asynchronous notifications to the bot's maintainers, etc.).
- **Live suggestions for editors.** It could also be possible to prompt some alerts or propose some content when a human editor starts working on a page whose abstract does not match with its class shape.
- **Automatic type suggestions affecting DBpedia.** Let us picture an abstract which uses all of the properties indicated in its corresponding shape, but still does not conform with its shape due to node constraint issues. For example, a band abstract could be linked with a record label, but that record label has not been typed with `dbo:RecordLabel` in DBpedia. Detecting such a constraint violation can produce typing suggestions for the mentioned entities in the target abstract.

5.5.1.3 Vandalism detection

Vandalism detection in crowd-sourced projects such as Wikipedia has attracted the interest of the research community[304, 305]. Automatically extracted shapes could be used as a tool to detect structural anomalies indicating potential vandalism.

This could be achieved by using SEITMA to produce shapes of vandalized entities. The shapes could be extracted by using target entities of different classes, so the resulting shapes contain vandalism features of non-specific knowledge domains. It could be also possible to extract vandalized shapes per class or using any other entity-clustering strategy. This way, the shapes produced would contain frequent vandal actions for a certain domain knowledge. In both cases, the shapes obtained should be compared to shapes from non-vandalized versions of the targets abstracts, so one can detect which are the actual constraints that only fit with the altered texts.

A shape-based system to detect vandalism could be able to discover structural signs of vandalism, such as presence of suspicious properties or usage of suspicious object types in relations, which could point to elements frequently used in vandal actions. For example, such elements may include content heavily unrelated to the target domain knowledge or references to trendy jokes.

A decrease of the general conformance of an abstract with its target shape after an edit could also be a sign of vandalism. This could be checked by running a triple

²⁹Discussion pages, also known as *talk pages*, are spaces in which Wikipedia editors can discuss modifications of a certain Wikipedia page. Each Wikipedia page has an associated discussion page. Read more at https://en.wikipedia.org/wiki/Help:Talk_pages Accessed in 2022/05/03.

extraction process before and after an edit, and trying to validate those triples using a previously extracted shape. If the second validation reveals more constraint violations than the first one, then the edit could be malicious.

5.5.2 Use cases out of Wikipedia

The use cases described above are mainly focused on improving the maintainability of Wikipedia, either by improving the editors experience or by automatically detecting malicious editing behavior. Some of the applications already mentioned for the Wikipedia scenario could be implemented also in any platform sharing this crowd-sourced nature. Examples of such projects can be other on-line encyclopedias, specialized wikis, on-line dictionaries, etc.

The challenges that should be tackled to run SEITMA on those sources are mainly linked to the used vocabularies. In special, the approach implemented in SEITMA-L is heavily dependent on the DBpedia ontology, which is tightly related to Wikipedia. SEITMA-F may need less adaptations to be used on sources with a similar structure to Wikipedia. However, in order to use SEITMA-F for any of the use cases described in section 5.5.1, it would be necessary to perform a study on how to adapt the FRED's output to each specific use case.

In the following subsections, we will mention and discuss potential uses on sources whose main purpose or essential structure is not that to Wikipedia.

5.5.2.1 Text validation and class summarization in formal environments

Any source composed of text descriptions which must fulfill some requirements may require text validation. Legal, technical, or scientific sources in formal environments are examples of such scenarios. SEITMA could be used then to extract shapes from some exemplary documents. The shapes automatically extracted could be used as drafts that may be tuned by some domain expert.

Once the shapes are validated by the domain experts, a new stage of triple extraction could be performed over the documents. In case the triples obtained from a document do not conform with its corresponding shape, the constraint violations can reveal lack of specific pieces of information in the document.

An alternative use for those shapes validated by domain experts would be the automatic generation of summaries which describe the fundamental parts of a document. This would require mapping a shape TC's to natural language sentences. However, unlike the generation of Wikipedia templates, this translation is not oriented to the generation of a template which needs to be completed with instance values, but to produce a textual expression of the TCs themselves.

Needless to say, those automatically extracted shapes could also be used just as they are. Its formal syntax can be useful to describe the features of a certain group of documents from a structural point of view.

5.5.2.2 Automatic types or tags

Many text-oriented platforms rely on the ability to make an effective classification of the different pieces of information that they contain. Tag systems in blogs and forums, categories in news portals, or entity classifications in encyclopedic platforms are examples of such useful classifications.

An implementation of SEITMA could be used in those systems to produce automatic text classification w.r.t. the entities described in the different text pieces. First,

SEITMA should be used to extract some model shapes using enough exemplary entities. Then, the triple extraction process should be performed over the target texts.

The classification could be produced at text level or at entity level. Text-level tagging would be produced when a certain target text piece conforms with a certain shape associated to a topic. Instance-level tagging would occur when some of the entities mentioned in the text produce a match with a shape associated to a certain class.

The automatic classification of text pieces is a well-studied problem [306]. Most of the modern approaches to perform automatic text classification are based on ML techniques. A usual problem with ML classifiers is that they work in a black-box manner. The user knows which are the algorithms used and may have an intuition on which are the determinant features to produce some classification results. However, most of the times, the decisions taken by ML algorithms are opaque.

The actual performance of using shapes in classification scenarios as the described ones still needs to be studied. However, an extra value that such an approach can bring is that decisions would be no longer black-box like. When a certain tag t is produced, the user can know which are the text features that caused t to be proposed from the TCs of the shape associated to t .

5.5.2.3 Application in text streams

Platforms such as Twitter, Reddit, or even some forums, can be viewed as text streams consisting of small actions of many different users posting about a wide variety of topics. The user of those platforms, especially Twitter, tend to post about trendy topics or events. A large amount of data is produced about such events in short time periods and, frequently, those topics stop being trendy and relevant for the community in short time periods too.

In such environments, the text stream produced by the community could be processed to produce triples. Simultaneously, when an entity e of such stream is detected to be an instance of a class c , the accumulated knowledge about e could be processed to generate or modify a shape associated to c .

Those shapes could be useful to produce automatic live classifications. Once a shape is already available and consolidated, the process of automatic triple extraction can still be active not to look for new shapes, but to recognize structures which lead to automatic typings for an entity. Again, performing such automatic classifications is already possible, but most of those approaches are based on ML techniques. As already mentioned, introducing shape languages in this scenario cause that the classifications produced less opaque.

The shapes used for this purpose could be generated in batch-processes with example data manually chosen instead of extracted from the actual stream data. With this, a system can be trained to discover instances of target classes (music releases, sport events, financial news, etc.). This can be already performed by heuristic-based approaches, such as tracking updates about specific tags or keywords. Those approaches allow to locate target pieces of information. However, using a triple extraction process over the text would also allow for performing automatic computations over specific pieces of information associated to the target events.

The heuristic-based approaches could be used as a first filter, so SEITMA is used just to parse text pieces which are found to be relevant because they contain some tags or keywords.

5.6 Related work

To the best of our knowledge, the only alternative to perform shape extraction from natural language is described in [22]. This approach is designed to parse conceptual descriptions which are indeed very similar to actual descriptions of shapes. The system can parse sentences such as “*Every user has exactly one family name*” or “*Each user has at least one contact mail*”. The text patterns recognized by this tool are described in [307]. We think that this type of tool may not be suitable to process social media content, where explicit conceptual descriptions may not be frequent.

The work described in [22] consists in a specific form of *model generation* based on textual requirements in which the obtained products are SHACL shapes. Some other approaches of model generation from text requirements extract different final products, such as Unified Modeling Language (UML) diagrams [308], or Entity-Relationship (ER) models [309, 310]. Most of the existing approaches combine general NLP techniques with domain-specific patterns and heuristics. Such patterns allow for detecting key pieces of knowledge that are mapped to some structured representation format. Even if some of the approaches accept unrestricted text as input [311, 312], the type of knowledge obtained is usually limited by the heuristics and patterns used.

The SEITMA approach is not a case of model extraction from requirements, but model extraction through generalization of examples. SEITMA propose two main tasks. Extraction of triples from natural language, and extraction of shapes from RDF graphs. System related to the later subtask has already been reviewed in section 4.5. In this section, we will make an overview of works related with the extraction of RDF (or RDF compatible models) from natural language. We will use the term *machine reader* to refer to those systems³⁰.

Many approaches have been proposed to solve subtasks of machine reading (NER, EL, POS, WSD, RE, etc.). Indeed, most of the machine readers consist of compilations of several systems with different roles working together.

Several machine readers are somehow related to Wikipedia corpora. Such approaches would be candidates to be integrated in a SEITMA implementation that, as with our current prototypes, can be used to extract shapes from a Wikipedia text corpus. Sometimes, the relation with Wikipedia is caused by a data dependency. Approaches in this situation use some ML methods that, same as LAREWA, are based on distant supervision. The systems described in [313] and [314] have such dependency. However, those approaches could be adapted to work with other sources, as long as it is possible to gather adequate training data with distant supervision. An example of distant supervision out of the Wikipedia context is described in [315]. The authors are able to annotate entities and relations between them in a text corpus from the New York Times using Freebase as a knowledge base.

Some other systems are just related with DBpedia because its published evaluation only utilizes Wikipedia data. Such systems do not rely on specific Wikipedia features and, theoretically, it could be easier to adapt them to other sources. This is the case of Graphia [316], Refractive [317], the approach described in [318], or BOA [319]. This last system was also evaluated using the news corpus described in [315].

Among existing machine readers successfully evaluated in open domains, apart from FRED, we can mention systems such as LODifier [281], PIKES [283], the techniques associated to Knowledge Vault [286] and DeepDive [320, 321], or the systems

³⁰Machine readers do not necessarily produce RDF outputs.

described in [284] and [282]. Those approaches, same as FRED, could be candidates to be integrated in a SEITMA implementation with unspecific target sources.

LODifier [281] is a system which implements a pipeline similar to FRED's one. It transforms the output of a deep semantic analysis performed with Boxer into RDF triples. LODifier also performs WSD with UKB to introduce mappings to WordNet, and EL with Wikifier [276] to link NEs with Wikipedia links. The text is processed with the statistical parser C&C, which is based on the CCGBank corpus [322]. This parser is used after applying a standard process of tokenization and detecting named entities with Wikifier.

DeepDive [320, 321] was³¹ a project to perform Knowledge Base Population (KBP) by integrating several different types of sources. Its NLP pipeline consisted of a combination of NER using StanfordNER [274], EL using a set of ad-hoc rules and heuristics to create links with Wikipedia pages (string matching, Wikipedia redirects, Google and Bing search results, etc.), and RE using the methods described in [278]. DeepDive proved the feasibility of performing large-scale KBP using unstructured web resources. The candidate mentions and relations obtained with the NLP pipeline were evaluated using statistical inference in Markov Logic. The necessary classifiers were trained using distant supervision supported by Freebase data. This approach was scaled using Condor [323] to perform parallel tasks.

Knowledge Vault [286] is another example of KBP, in this case using different types of sources. Knowledge Vault include knowledge extracted from natural language, but also from other types of semi-structured sources, such as tables or HTML pages. Several extractors can work on information represented in different manners. The triples stored in Knowledge Vault have an associated confidence score in $[0, 1]$, where 1 means maximum confidence. This is based on agreement between different extractors. The KG described by the authors contains 1.6B triples, from which 271M triples has a confidence level above 0.9. This work implements distant supervision using Freebase knowledge. The authors introduce the notion of *local closed world assumption* which is also used in LAREWA.

PIKES is a frame-based system for ontology population. Its workflow consists of two main stages. During the first one, standard NLP techniques are performed. In this stage, CoreNLP [293] is used to perform the first parsing tasks (tokenization, POS, lemmatization, etc.). WSD is implemented with UKB, and DBpedia Spotlight [125] is used to link mentions with DBpedia URIs. Also, two different tools are used to perform SRL: Semafor [324] w.r.t. FrameNet, and Mate-tools [325] w.r.t. PropBank [326, 327] and NomBank [328]. The second stage, named *knowledge distillation*, aims to produce a graph for which the Unique Name Assumption is met. That is, nodes which are detected to refer to the same entity are merged. With this, PIKES abstracts the knowledge obtained from the specific mentions to entities and relations in the input text. This stage is supported by the tool RDF_{PRO} [174], which allows for performing data filtering and transformation, RDFS inference, *owl:sameAs* smushing, and statistics extraction on RDF content.

Another machine reader is presented in [284]. The main novelty of this approach is the strategy to select an adequate property of a certain KB in order to represent the relation between two entities. Binary relations between entities are extracted using OpenIE [329]. Then, candidate relations between pairs of entities are selected using a SPARQL-based approach. The candidate property chosen is the one which is more similar to the binary relation previously extracted. The similarity is computed using

³¹It is not under active development since 2017, as stated in <http://deepdive.stanford.edu/> Accessed in 2022/05/03.

a *knowledge-based* score obtained using notions of semantic sources such as WordNet, and a *corpus-based* score obtained using word embeddings. The authors implement a prototype which integrates this idea using standard tools for others tasks: CoreNLP for general preprocessing and POS; DBpedia Spotlight, TagME, Babelify [277], and WAT [330] for NEL; ClausIE [331] for SRL.

In [282], an approach to support distant supervised systems is presented. To improve the precision of those systems, the authors propose to filter ambiguous seed nodes used to train classifiers. For example, lexicalizations with too many potential senses are discarded as seed node candidates. In order to improve the recall, they propose a scope broader than a sentence to find relations between entities. The authors argue that paragraphs may contain constant subjects and objects. This is, if two entities with a known relation r are mentioned in different sentences of the same paragraph, then the paragraph may be a valid expression of the relation r . With such approach, finding actual relations between entities requires an especial emphasis on CRR to detect pronouns or synonyms of entities involved in known relations. CRR and other usual NLP-related tasks of the system implemented in [282] are performed with CoreNLP.

To assist distant supervision techniques, *sar-graphs* are proposed in [332]. *Sar-graphs* are meant to be a bridge between linguistic sources such as WordNet and knowledge graphs such as DBpedia or Wikidata. *Sar-graphs* link semantic relations from factual knowledge graphs with their linguistic representations in human language. These graphs contain linguistic constructions to represent semantic relations in specific languages. Vertexes in such constructions are words that can be related with data such as word form, lemma, word class, word sense, global identifiers, or some statistical metadata. The constructions (extraction patterns) are modeled as sub-trees of dependency-graph representations of sentences. To prove the feasibility of their approach, the authors build *sar-graphs* for 25 different relations using off-the-shell tools for NLP-related tasks.

Extended reviews about techniques and tools used to represent natural language using semantic web standards are available in [24] and [333].

5.7 Conclusions

At the beginning of this chapter, we stated the following Research Question:

- **RQ1:** How can we automatically extract shapes from social media content?

In this chapter, we have described SEITMA, an architecture which combines several subsystems to perform automatic extraction of shapes from natural language pieces.

We have implemented two different prototypes following SEITMA specifications. Both prototypes were able to extract shapes describing classes by mining Wikipedia abstracts. The shapes obtained using these prototypes prove the feasibility of SEITMA proposals in order to tackle the challenges associated with **RQ1**.

We have performed an experiment to evaluate the potential of both prototypes: we used them to extract shapes associated to the most important classes of DBpedia using Wikipedia abstracts. Both prototypes achieved promising results, but the experiments revealed ways in which the performance of both systems could be improved.

The prototype named SEITMA-L implements an ML approach based on distant supervision which uses Wikipedia and DBpedia data. The shapes obtained with

SEITMA-L under our experimental conditions were concise, based on the vocabulary of the *dbo* ontology, and easy to interpret. However, the data obtained with distant supervision caused that many relevant properties were not included in the resulting shapes, as it was not possible to train reliable ML classifiers for that. Our hypothesis is that SEITMA-L can produce better results when it is used with human annotated data, even if this may affect the system's adaptability to different contexts.

The prototype named SEITMA-F was able to produce shapes with a high level of detail using several vocabularies. We detected that the amount and variety of features generated under the conditions of our experiments could be even excessive. We proposed some alternative settings and prototype adaptations to filter noisy features and to increase the specificity of the shapes obtained. Nevertheless, the exact nature of such modifications may be tightly related to specific contexts of application.

We have introduced several use cases in which using a SEITMA implementation could be beneficial. Some of the cases refer to actions that could have a positive impact on Wikipedia. The goals described in those cases could be achieved with minor adaptations of our current SEITMA implementations. Other use cases describe potential uses of SEITMA out of the Wikipedia context. This last group of use cases may be achieved with customized versions of SEITMA-F or using a new SEITMA implementation whose module for triple extraction is adapted to the features of the target text corpus (or text stream).

5.7.1 Future work

The experiments performed in this chapter allowed us to detect several lines of future work:

- **Experiments with SEITMA-L using human-annotated training data.** Many properties that a human annotator could have identified in the target abstracts were not included in the shapes produced by SEITMA-L because there was not enough training data. We think that the variety of features observed in SEITMA-L shapes could be increased by using human-annotated examples. An experiment using human annotated data would probably require to choose some specific classes/domain knowledges to be able to produce enough training data.
- **Full implementation and evaluation of use cases.** Most of the use cases introduced in this chapter use the shapes produced by SEITMA as a source of knowledge. However, they require some other systems to solve their respective issues, such as automatic classifiers, error detectors, or template generators. Those subsystems should receive shapes as input and be able to create some other outputs or perform some further actions with these shapes.
- **SEITMA implementation for working with noisy sources.** Such an implementation would need a triple extraction submodule able to deal with special features associated to this kind of context: lower quality writing, lack of context, jargon, etc. Such an implementation would allow to use SEITMA for extracting shapes from social platforms such as Twitter or Reddit.

Chapter 6

Conclusions

6.1 Conclusions (English version)

We have developed a number of solutions to provide answers to three different Research Questions. In this chapter, we propose answers for each of those questions based on results described in chapters 3, 4, and 5.

RQ3 - How can we identify the most important classes of an RDF graph?

A feasible and top performant way of doing that is applying ClassRank. ClassRank is an algorithm that determines the importance of a class within an RDF graph. Each class is assigned a score which is the sum of the PageRank scores of its instances.

We compared ClassRank with several state-of-the-art approaches to measure class importance in different KGs. We built reference rankings of class importance for each source using SPARQL logs. In those rankings, the position of a class is determined by how frequently it appears in the logs. The different techniques were evaluated by determining the similarity between the importance ranking they produced and the reference rankings. That similarity was calculated using Ranking Biased Overlap.

The best-performing technique was ClassRank. Surprisingly, the technique producing the closest results to ClassRank was Instance Counting. This approach determines the importance of a class by simply counting how many instances it has. The main difference observed between rankings produced by ClassRank and Instance Counting is that the later one penalizes classes that have few instances (e.g., *Country*) even when those instances (e.g., *Japan*, *India*, *Australia*) are fundamental for the graph's structure. In contrast, such classes tend to be high-ranked with ClassRank.

The techniques evaluated can be classified in two groups: those only using conceptual knowledge (OTT techniques) and those which also use instance-level knowledge (AAT techniques). Our study also revealed that, in general, AAT approaches perform better than OTT ones.

Cf. chapter 3 for further details.

RQ2 - How can we produce shapes by mining RDF triples?

A feasible way to do that is using sheXer. sheXer is an automatic instance-based shape extractor. It uses a voting system which can detect features frequently observed among a group of entities, so they can be generalized into a shape. sheXer implements an iterative approach that avoids loading into main memory any content that is not relevant for the execution. This allows for processing large datasets with inexpensive hardware.

When we start working in problems related to **RQ2**, the automatic extraction of shapes was a barely explored research field, as shape languages were a novelty at the time. sheXer was one of the first available systems to perform such a task. Nowadays, a number of alternatives have been proposed. Still, sheXer is a competitive solution, as it has a unique combination of features among the existing systems:

- It can generate both ShEx and SHACL.
- It can handle large real-world datasets.
- The inputs can be provided in several ways, such as local RDF files or remote SPARQL endpoints.
- It allows for filtering shape features which are not frequent among the relevant instances.
- It performs shape inter-linkage, i.e., it produces shapes that can reference to each other.
- It allows for customizing the extraction process with many different options.

Although sheXer can handle large datasets, our experiments with sheXer reveal linear relations between memory usage and some input variables, such as the number of target shapes or the number of relevant entities. Such relations can be a scalability issue when processing too large datasets. Yet, we have shown that when sheXer is fed with a reduced but representative set of instances the memory usage is drastically reduced and the shapes obtained are not noticeably affected.

Cf. chapter 4 for further details.

RQ1 - How can we automatically extract shapes from social media content?

A feasible way to do that is using a SEITMA implementation. SEITMA is an architecture that describes how to combine subsystems of triple extraction from natural language and shape extraction from RDF. It is based on example generalization. SEITMA inputs are expected to contain descriptions or statements about several instances of some concepts. As output, it produces shapes associated to those concepts containing the features most frequently occurring in the examples.

To the best of our knowledge, there is just another proposal to obtain shapes from natural language apart from SEITMA. However, such system is designed to work with texts including actual descriptions of shape features, a kind of content that could be hard to find in social media.

We implemented two prototypes of SEITMA. Both use sheXer to extract shapes due to the following reasons:

- It can produce ShEx and SHACL content, which may allow SEITMA to reach a broader set of developers and practitioners.
- It can extract shapes with inverse paths, i.e., shapes where the focus node is used as the object of a triple. This allows for producing more constraints associated to the shapes obtained.
- It generates textual comments attached to the shape's constraints which are valuable knowledge and can be used as input further automatic processes.
- It can handle large RDF datasets.

We performed a similar experiment with both prototypes: extracting shapes associated to the most important classes in DBpedia using Wikipedia abstracts. The most important classes of DBpedia were determined using ClassRank. The most important entities for those classes were used as model to train Machine Learning algorithms for one of the prototypes. The other prototype relies on general Language Models independent from Wikipedia.

Both prototypes were able to extract promising but different shapes using the input abstracts. While one of the prototypes produced concise shapes using vocabulary associated to DBpedia, the other one produced shapes with many details built on top of several vocabularies. We discussed ways to adapt these systems so they can be used in real scenarios. Such adaptations range from gathering different training data to using context-dependent graph mappings.

We also described several use cases where using shapes produced by a SEITMA implementation could be helpful. We discussed potential implementations or adaptations of our current prototypes to be used in such use cases.

Cf. chapter 5 for further details.

General conclusions - We have been able to provide an answer for each of the three Research Questions posed in this work. To do it, we have developed three different systems: the ClassRank algorithm, the sheXer library, and the SEITMA architecture.

Even if ClassRank and sheXer were developed to satisfy the needs of SEITMA, they are separate tools which can be used in other contexts apart from those considered in this thesis. We have produced several publications where these two systems are described or evaluated as standalone items [199, 252–254]. In particular, sheXer seems to have been especially well received by the scientific community. It has already been included as a piece of other scientific works [261, 262], and integrated in WikidataIntegrator¹, which is a relevant tool for the Wikidata community. It has also been positively evaluated in a recent comparison among different solutions for automatic shape extraction [25].

While our current two SEITMA prototypes use sheXer and can benefit from ClassRank, the SEITMA architecture is independent of these two tools.

We have outlined several lines of future work for ClassRank, sheXer, and SEITMA. For example, we want to evaluate ClassRank as a relevance metric instead of an importance one. Also, we would like to implement and evaluate new versions of ClassRank based on notions of entity importance different from PageRank. W.r.t. sheXer, we want to add several functional features to the system, including ontology processing to detect and produce shape hierarchies. We also aim to implement and evaluate optimizations of our library to tackle scalability issues when working with too large input data, and to produce adaptations of our library to work with KG models different to RDF. Finally, our priority with SEITMA is to use an implementation to solve some of the discussed use cases.

Those lines of research will be developed in parallel over the next few years.

6.2 Conclusiones (Versión en castellano)

Hemos desarrollado varias soluciones que dan respuesta a las Preguntas de Investigación planteadas. En este capítulo, respondemos a dichas preguntas en base a los resultados descritos en los capítulos 3, 4 y 5.

¹<https://github.com/SuLab/WikidataIntegrator> Accessed in 2022/05/03.

RQ3 - ¿Cómo podemos identificar las partes más importantes de un grafo RDF?

Una forma factible y de eficacia probada para hacerlo es el uso de ClassRank. ClassRank es un algoritmo que mide la importancia de clases en grafos RDF. A cada clase se le asigna una puntuación calculada como la suma de la importancia de sus instancias. La importancia a nivel de instancia se mide usando PageRank.

Hemos comparado ClassRank con varias alternativas existentes para medir importancia de clases en grafos de conocimiento. Para ello, hemos elaborado unos rankings de importancia basados en logs de SPARQL. En estos rankings, la posición de cada clase se determina en función de la frecuencia con la que es mencionada en los logs. Las diferentes técnicas son evaluadas midiendo la similitud entre los rankings que producen con los rankings de referencia basados en logs. Dicha similitud se calcula usando *Ranking Biased Overlap*.

ClassRank fue la técnica que obtuvo mejores resultados durante nuestra experimentación. Sorprendentemente, la técnica que produjo resultados más cercanos a ClassRank fue Conteo de Instancias. Dicha métrica consiste en asignar a cada clase una importancia basada simplemente en su número de instancias declaradas en el grafo. La mayor diferencia observada entre los rankings de ClassRank y Conteo de Instancias es que esta última métrica penaliza aquellas clases que tienen pocas instancias, aun cuando estas resultan fundamentales para la estructura del grafo. Ejemplo de ello sería la clase *País*, cuyas instancias (*Japón, India, Australia, etc.*) actúan como puentes entre distintas regiones de los grafos evaluados. En cambio, tales clases aparecen en las posiciones más altas del ranking usando ClassRank.

Las técnicas evaluadas pueden ser divididas en dos grandes grupos: aquellas que solo utilizan conocimiento conceptual (llamadas *OTT*) y aquellas que además utilizan conocimiento a nivel de instancia (llamadas *AAT*). Nuestro estudio también revela que, en general, las métricas *AAT* producen rankings de mayor calidad que las *OTT*.

En el capítulo 3 se ofrecen más detalles a este respecto.

RQ2 - ¿Cómo podemos producir shapes mediante minado de tripletas RDF?

Una forma factible de lograrlo es el uso de sheXer. sheXer es un extractor automático de shapes que utiliza minado de conocimiento a nivel de instancia. Implementa un sistema de votos con el cual es capaz de detectar características topológicas observadas con frecuencia entre un grupo de instancias. Esto permite generalizar tales características para producir shapes. sheXer implementa un flujo iterativo que evita mantener en memoria principal aquel contenido que no es relevante para el proceso en ejecución. De esta manera es capaz de procesar grandes volúmenes de datos utilizando hardware asequible.

Cuando comenzamos a trabajar en temas relacionados con la pregunta **RQ2**, la extracción automática de shapes era un campo de investigación prácticamente inexplorado, pues los lenguajes de shapes eran relativamente nuevos. sheXer fue uno de los primeros sistemas disponibles para llevar a cabo esta tarea. A día de hoy, sheXer continúa siendo un sistema competitivo, pues exhibe una combinación de características que no existe entre los sistemas alternativos:

- Puede generar tanto ShEx como SHACL.
- Puede procesar grandes grafos RDF.
- Permite filtrar aspectos de shapes que no son vistos con la suficiente frecuencia entre las instancias minadas.

- Soporta referencias entre shapes, es decir, la etiqueta de una shape puede intervenir en la definición de otra shape.
- Permite personalizar el proceso de extracción a través de múltiples opciones de configuración.

Aunque sheXer ha probado ser capaz de procesar grandes volúmenes de datos, nuestros experimentos revelan relaciones lineales entre el consumo de memoria y algunas características de los datos de entrada, como el número de instancias relevantes para el proceso. Estas relaciones pueden causar problemas de escalabilidad con fuentes demasiado grandes. No obstante, hemos demostrado que la calidad de las shapes producidas por sheXer es similar cuando se utiliza un conjunto representativo de instancias y cuando se utilizan todas las instancias disponibles. Cuando se usa un conjunto reducido de instancias, el consumo de memoria mejora de forma drástica.

En el capítulo 4 se ofrecen más detalles a este respecto.

RQ1 - ¿Como podemos extraer shapes a partir de contenido textual generado en medios sociales?

Una forma factible de lograrlo es el uso de implementaciones de SEITMA. SEITMA es una arquitectura que describe como combinar procesos de extracción de tripletas a partir de lenguaje natural y de shapes a partir de contenido RDF. La arquitectura se basa en la generalización de ejemplos. La entrada esperada para la arquitectura es un corpus que contiene descripciones de varias instancias de ciertos conceptos. El sistema produce shapes asociadas a dichos conceptos que contienen las características más observadas entre sus instancias.

Hasta donde tenemos constancia, aparte de SEITMA, solo existe otro sistema capaz de generar shapes a partir de contenido textual. No obstante, dicho sistema está diseñado para procesar textos incluyendo descripciones estructurales de conceptos. Estimamos que ese tipo de contenido es más infrecuente en redes sociales que el conocimiento expresado a nivel de instancia.

Hemos implementado dos prototipos de SEITMA. Ambos prototipos utilizan sheXer para la extracción de shapes a partir de RDF debido a las siguientes razones:

- Puede producir contenido tanto en ShEx como en SHACL, lo que nos puede permitir alcanzar una audiencia más amplia para nuestro sistema.
- Puede extraer shapes con rutas inversas, es decir, shapes en las que el *nodo foco* actúa como objeto en una tripleta. Esto aumenta la información potencial que podemos añadir a cada shape.
- Puede generar comentarios asociados a las shapes producidas. Dichos comentarios, son una fuente de información valiosa. Aunque los comentarios no tienen una estructura formal estricta, su patrón es predecible y estable, lo que permite que procesos automáticos puedan parsear esta información.
- Puede manejar grandes volúmenes de datos.

Hemos llevado a cabo un experimento similar con ambos prototipos: hemos extraído shapes asociadas a las clases más importantes de DBpedia usando como fuente resúmenes de Wikipedia. Las clases más importantes de DBpedia fueron determinadas mediante el algoritmo ClassRank. Las entidades más importantes de dichas clases fueron usadas como datos modelo para uno de los prototipos, que se

basa en un sistema de *Machine Learning*. El otro prototipo utiliza modelos lingüísticos independientes de Wikipedia.

Si bien las shapes generadas por estos sistemas tienen distintas características, ambos fueron capaces de producir resultados prometedores. Mientras que uno de los prototipos generó shapes concisas usando vocabulario de DBpedia, el otro generó estructuras con un alto nivel de detalle que usan vocabularios tanto de DBpedia como de otras fuentes. Hemos discutido diferentes formas de adaptar estos prototipos para su uso en escenarios reales. Tales adaptaciones van desde una aproximación diferente para la obtención de datos modelo hasta la utilización de mapeos de grafo dependientes del contexto de aplicación.

También hemos descrito diferentes casos de uso en los que la utilización de shapes producidas por una implementación de SEITMA pueden ser útiles. Hemos discutido potenciales implementaciones de nuestra arquitectura o posibles adaptaciones de nuestros prototipos para ser utilizados en estos casos.

En el capítulo 5 se ofrecen más detalles a este respecto.

Conclusiones generales - Hemos sido capaces de dar respuesta a las tres Preguntas de Investigación planteadas. Para ello, hemos desarrollado tres principales propuestas distintas: el algoritmo ClassRank, el sistema sheXer y la arquitectura SEITMA.

A pesar de que ClassRank y sheXer fueron desarrollados para resolver distintas necesidades relacionadas con SEITMA, son productos que pueden ser utilizados en contextos distintos al de nuestra tesis. Hemos producido varias publicaciones en las que estos dos sistemas son descritos y evaluados como elementos independientes [199, 252–254]. Particularmente, sheXer parece haber sido bien recibido por la comunidad científica. Ya ha sido incluido como parte de otros trabajos [261, 262] y también ha sido integrado en WikidataIntegrator², una herramienta relevante para la comunidad de Wikidata. También ha sido evaluado de forma positiva en una comparación reciente de herramientas para la extracción automática de shapes [25].

A pesar de que nuestros prototipos de SEITMA utilizan sheXer y ClassRank, la definición de la arquitectura también es independiente de estos dos elementos.

Hemos descrito varias líneas de trabajo futuro relacionadas con ClassRank, sheXer y SEITMA. Por ejemplo, planeamos evaluar ClassRank como métrica de relevancia en lugar de métrica de importancia. Además, planteamos implementar y evaluar diferentes versiones de ClassRank basadas en nociones de importancia de instancia distintas a PageRank. Con respecto a sheXer, hemos planteado añadir varias características funcionales a nuestro sistema. Además, buscamos implementar y evaluar adaptaciones de la librería para lidiar con potenciales problemas de escalabilidad. Buscamos también implementar versiones de sheXer capaces de extraer estructuras de grafos de conocimiento con modelos no basados en RDF. Por último, nuestra prioridad respecto a SEITMA es la experimentación con un prototipo para resolver casos de usos reales.

Estas líneas de trabajo futuro serán desarrolladas en paralelo en los próximos años.

²<https://github.com/SuLab/WikidataIntegrator> Consultado por última vez el 03/05/2022.

Appendix A

Prefix definitions

PREFIX : <http://example.org/>
PREFIX boxer : <http://www.ontologydesignpatterns.org/ont/boxer/boxer.owl#>
PREFIX boxing : <http://www.ontologydesignpatterns.org/ont/boxer/boxing.owl#>
PREFIX dul : <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#>
PREFIX dbo : <http://dbpedia.org/ontology/>
PREFIX dbr : <http://dbpedia.org/resource/>
PREFIX ex : <http://example.org/>
PREFIX fe : <http://www.ontologydesignpatterns.org/ont/framenet/abox/fe/>
PREFIX foaf : <http://xmlns.com/foaf/0.1/>
PREFIX framester : <http://w3id.org/framester/schema/>
PREFIX fred : <http://www.ontologydesignpatterns.org/ont/fred/domain.owl#>
PREFIX geo : <http://www.opengis.net/ont/geosparql#>
PREFIX org : <http://www.w3.org/ns/org#>
PREFIX p : <http://www.wikidata.org/prop/>
PREFIX pq : <http://www.wikidata.org/prop/qualifier/>
PREFIX ps : <http://www.wikidata.org/prop/statement/>
PREFIX rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs : <http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema : <http://schema.org/>
PREFIX sh : <http://www.w3.org/ns/shacl#>
PREFIX skos : <http://www.w3.org/2004/02/skos/core#>
PREFIX sx : <http://shex.io/ns/shex#>
PREFIX wd : <http://www.wikidata.org/entity/>
PREFIX wdt : <http://www.wikidata.org/prop/direct/>
PREFIX wdt:n : <http://www.wikidata.org/prop/direct-normalized/>
PREFIX weso-s : <http://weso.es/shapes/>
PREFIX wikibase : <http://wikiba.se/ontology#>
PREFIX xml : <http://www.w3.org/XML/1998/namespace/>
PREFIX xsd : <http://www.w3.org/2001/XMLSchema#>

Bibliography

- [1] Timothy J Berners-Lee. *Information management: A proposal*. Tech. rep. CERN, 1989.
- [2] Lawrence Page et al. "The PageRank citation ranking: bringing order to the web." In: *Stanford InfoLab* (1999).
- [3] Tim O'reilly. *What is web 2.0*. " O'Reilly Media, Inc.", 2009.
- [4] Yunis Ali Ahmed et al. "Social media for knowledge-sharing: A systematic literature review". In: *Telematics and Informatics* 37 (2019), pp. 72–112. ISSN: 0736-5853. DOI: <https://doi.org/10.1016/j.tele.2018.01.015>. URL: <https://www.sciencedirect.com/science/article/pii/S0736585317306688>.
- [5] Lincoln Dahlberg. "Which social media? A call for contextualization". In: *Social Media+ Society* 1.1 (2015).
- [6] Nancy K Baym. "Social media and the struggle for society". In: *Social Media+ Society* 1.1 (2015).
- [7] Daniel Gayo-Avello. "Social media: a definition". In: *Personal Blog* (2022).
- [8] Tracy L Tuten and Michael R Solomon. *Social media marketing*. Sage, 2017.
- [9] Christopher Ifeanyi Eke et al. "A survey of user profiling: State-of-the-art, challenges, and solutions". In: *IEEE Access* 7 (2019), pp. 144907–144924.
- [10] Punam Bedi and Chhavi Sharma. "Community detection in social networks". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 6.3 (2016), pp. 115–135.
- [11] Daniel Gayo-Avello. "A meta-analysis of state-of-the-art electoral prediction from Twitter data". In: *Social Science Computer Review* 31.6 (2013), pp. 649–679.
- [12] Sharath Chandra Guntuku et al. "Detecting depression and mental illness on social media: an integrative review". In: *Current Opinion in Behavioral Sciences* 18 (2017), pp. 43–49.
- [13] Amaia Eskisabel-Azpiazu, Rebeca Cerezo-Menéndez, and Daniel Gayo-Avello. "An ethical inquiry into youth suicide prevention using social media mining". In: *Internet Research Ethics for the Social Age* 227 (2017).
- [14] J Brian Houston et al. "Social media and disasters: a functional framework for social media use in disaster planning, response, and research". In: *Disasters* 39.1 (2015), pp. 1–22.
- [15] Sally M Gainsbury et al. "An exploratory study of gambling operators' use of social media and the latent messages conveyed". In: *Journal of Gambling Studies* 32.1 (2016), pp. 125–141.
- [16] John Hani et al. "Social media cyberbullying detection using machine learning". In: *Int. J. Adv. Comput. Sci. Appl* 10.5 (2019), pp. 703–707.
- [17] Kai Shu et al. "Fake news detection on social media: A data mining perspective". In: *ACM SIGKDD explorations newsletter* 19.1 (2017), pp. 22–36.

- [18] Daniel A González-Padilla and Leonardo Tortolero-Blanco. "Social media influence in the COVID-19 pandemic". In: *International braz j urol* 46 (2020), pp. 120–124.
- [19] Roy Fielding et al. *RFC2616: Hypertext Transfer Protocol–HTTP/1.1*. 1999.
- [20] Holger Knublauch and Dimitris Kontokostas. "Shapes constraint language (SHACL)". In: *W3C Candidate Recommendation* 11.8 (2017).
- [21] Eric Prud'hommeaux, Jose Emilio Labra Gayo, and Harold Solbrig. "Shape expressions: an RDF validation and transformation language". In: *Proceedings of the 10th International Conference on Semantic Systems*. 2014, pp. 32–40.
- [22] David Šenkýř. "SHACL Shapes Generation from Textual Documents". In: *Enterprise and Organizational Modeling and Simulation*. Ed. by Robert Pergl et al. Springer International Publishing, 2019, pp. 121–130. ISBN: 978-3-030-35646-0.
- [23] Razieh Baradaran, Razieh Ghiasi, and Hossein Amirkhani. "A survey on machine reading comprehension systems". In: *Natural Language Engineering* (2020), pp. 1–50.
- [24] Jose L Martinez-Rodriguez, Aidan Hogan, and Ivan Lopez-Arevalo. "Information extraction meets the semantic web: a survey". In: *Semantic Web* 11.2 (2020), pp. 255–335.
- [25] Kashif Rabbani, Matteo Lissandrini, and Katja Hose. "SHACL and ShEx in the Wild: A Community Survey on Validating Shapes Generation and Adoption". In: *Companion Proceedings of the Web Conference*. Ed. by Anna Lisa Gentile and Lisena Pasquale. WWW'22 Companion. ACM, 2022.
- [26] Denny Vrandečić and Markus Krötzsch. "Wikidata: a free collaborative knowledgebase". In: *Communications of the ACM* 57.10 (2014), pp. 78–85.
- [27] Jens Lehmann et al. "DBpedia—a large-scale, multilingual knowledge base extracted from Wikipedia". In: *Semantic Web* 6.2 (2015), pp. 167–195.
- [28] Alexandros Pappas et al. "Exploring importance measures for summarizing RDF/S KBs". In: *European Semantic Web Conference*. Springer. 2017, pp. 387–403.
- [29] Tim Berners-Lee, James Hendler, and Ora Lassila. "The semantic web". In: *Scientific American* 284.5 (2001), pp. 34–43.
- [30] Tim Berners-Lee, Roy Fielding, and Larry Masinter. *RFC2396: Uniform resource identifiers (URI): generic syntax*. 1998.
- [31] Christian Bizer, Tom Heath, and Tim Berners-Lee. "Linked data: The story so far". In: *Semantic services, interoperability and web applications: emerging concepts*. IGI global, 2011, pp. 205–227.
- [32] Dan Brickley, Ramanathan V Guha, and Brian McBride. "RDF Schema 1.1". In: *W3C recommendation* (2014).
- [33] Deborah L McGuinness, Frank Van Harmelen, et al. "OWL web ontology language overview". In: *W3C recommendation* (2004).
- [34] Kurt Bollacker et al. "Freebase: a collaboratively created graph database for structuring human knowledge". In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM. 2008, pp. 1247–1250.
- [35] Douglas B Lenat. "CYC: A large-scale investment in knowledge infrastructure". In: *Communications of the ACM* 38.11 (1995), pp. 33–38.

- [36] Barbara B Tillett. "A Virtual International Authority File." In: *ERIC*. URL: <https://eric.ed.gov/?id=ED459769> (2001).
- [37] Dan Brickley and Libby Miller. "FOAF Vocabulary Specification 0.99". In: *XMLNS* (2014).
- [38] Thomas Pellissier Tanon, Gerhard Weikum, and Fabian Suchanek. "Yago 4: A reason-able knowledge base". In: *European Semantic Web Conference*. Springer. 2020, pp. 583–596.
- [39] Andy Seaborne et al. "SPARQL 1.1 Federated Query". In: *W3C recommendation* (2013).
- [40] *Linked Data*. <https://www.w3.org/DesignIssues/LinkedData.html>. Accessed: 2022-05-09. 2009.
- [41] W3C. "RDF". In: *W3C recommendation* (2014).
- [42] Jose Emilio Labra Gayo et al. "Validating RDF data". In: *Synthesis Lectures on Semantic Web: Theory and Technology* 7.1 (2017), pp. 1–328.
- [43] Tim Berners-Lee and Dan Connolly. "Notation3 (N3): A readable RDF syntax". In: *W3C recommendation* (2011).
- [44] Eric Prud'hommeaux and Gavin Carothers. "RDF 1.1 Turtle". In: *W3C recommendation* (2014).
- [45] Gavin Carothers and Andy Seaborne. "RDF 1.1 N-Triples". In: *W3C recommendation* (2014).
- [46] Manu Sporny et al. "JSON-LD 1.0". In: *W3C recommendation* 16 (2014), p. 41.
- [47] Daniel Hernández, Aidan Hogan, and Markus Krötzsch. "Reifying RDF: What Works Well With Wikidata?" In: *SSWS@ ISWC 1457* (2015), pp. 32–47.
- [48] Jeremy J Carroll et al. "Named graphs". In: *Journal of Web Semantics* 3.4 (2005), pp. 247–267.
- [49] Edward W Schneider. "Course Modularization Applied: The Interface System and Its Implications For Sequence Control and Data Analysis". In: *ERIC*. URL: <https://eric.ed.gov/?id=ED088424> (1973).
- [50] Amit Singhal. *Introducing the Knowledge Graph: things, not strings*. Google Blog. 2012. URL: <https://www.blog.google/products/search/introducing-knowledge-graph-things-not/>.
- [51] Lisa Ehrlinger and Wolfram Wöß. "Towards a definition of knowledge graphs." In: *SEMANTiCS (Posters, Demos, SuCCESS)* 48.1-4 (2016), p. 2.
- [52] Piero Andrea Bonatti et al. "Knowledge Graphs: New Directions for Knowledge Representation on the Semantic Web". In: *Dagstuhl Seminar 18371*. Ed. by Piero Andrea Bonatti et al. Vol. 8. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, pp. 29–111.
- [53] Michael K. Bergman. *A Common Sense View of Knowledge Graphs*. Adaptive Information, Adaptive Innovation, Adaptive Infrastructure Blog. 2019. URL: <http://www.mkbergman.com/2244/a-common-sense-view-of-knowledge-graphs/>.
- [54] Aidan Hogan et al. *Knowledge Graphs*. English. Synthesis Lectures on Data, Semantics, and Knowledge 22. Morgan & Claypool, 2021. ISBN: 9781636392363.
- [55] Xiao Wang et al. "Heterogeneous graph attention network". In: *The world wide web conference*. 2019, pp. 2022–2032.

- [56] Justin J Miller. “Graph database applications and concepts with Neo4j”. In: *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*. Vol. 2324. 2013.
- [57] Renzo Angles, Harsh Thakkar, and Dominik Tomaszuk. “RDF and Property Graphs Interoperability: Status and Issues”. In: *AMW*. Aug. 2019.
- [58] R. J. Pittman et al. *Cracking the Code on Conversational Commerce*. eBay Blog. 2017. URL: <https://www.ebayinc.com/stories/news/cracking-the-code-on-conversational-commerce/>.
- [59] Natasha Noy et al. “Industry-scale Knowledge Graphs: Lessons and Challenges”. In: *ACM Queue* 17.2 (2019).
- [60] Arun Krishnan. *Making search easier: How Amazon’s Product Graph is helping customers find products more easily*. Amazon Blog. 2018. URL: <https://blog.aboutamazon.com/innovation/making-search-easier>.
- [61] Saurabh Shrivastava. *Bring rich knowledge of people, places, things and local businesses to your apps*. Bing Blogs. 2017. URL: <https://blogs.bing.com/search-quality-insights/2017-07/bring-rich-knowledge-of-people-places-things-and-local-businesses-to-your-apps>.
- [62] Ferras Hamad, Isaac Liu, and Xian Xing Zhang. *Food Discovery with Uber Eats: Building a Query Understanding Engine*. Uber Engineering Blog. 2018. URL: <https://eng.uber.com/uber-eats-query-understanding/>.
- [63] Gregory Karvounarakis et al. “RQL: A functional query language for RDF”. In: *The Functional Approach to Data Management*. Springer, 2004, pp. 435–465.
- [64] Michael Sintek and Stefan Decker. “TRIPLE—A query, inference, and transformation language for the semantic web”. In: *International semantic web conference*. Springer. 2002, pp. 364–378.
- [65] Sebastian Schaffert and François Bry. “Querying the Web Reconsidered: A Practical Introduction to Xcerpt.” In: *Extreme Markup Languages®*. 2004.
- [66] Eric Prud’hommeaux. “Algae RDF Query Language”. In: *W3C draft (work in progress)* (2004).
- [67] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. “Semantics and complexity of SPARQL”. In: *ACM Transactions on Database Systems (TODS)* 34.3 (2009), pp. 1–45.
- [68] Thomas R Gruber. “A translation approach to portable ontology specifications”. In: *Knowledge acquisition* 5.2 (1993), pp. 199–220.
- [69] Deborah L McGuinness and Frank van Harmelen. “OWL Web Ontology Language. Document Overview”. In: *W3C Recommendation* (2004).
- [70] W3C OWL Working Group. “OWL 2 Web Ontology Language. Document Overview (Second Edition)”. In: *W3C Recommendation* (2012).
- [71] Zhongli Ding and Yun Peng. “A probabilistic extension to ontology language OWL”. In: *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*. IEEE. 2004, 10–pp.
- [72] Jeff Z Pan and Ian Horrocks. “OWL FA: a metamodeling extension of OWL D”. In: *Proceedings of the 15th international conference on World Wide Web*. 2006, pp. 1065–1066.

- [73] David Martin et al. "Bringing semantics to web services: The OWL-S approach". In: *International Workshop on Semantic Web Services and Web Process Composition*. Springer. 2004, pp. 26–42.
- [74] Grégory Alary et al. "Comp-O: an OWL-S Extension for Composite Service Description". In: *International Conference on Knowledge Engineering and Knowledge Management*. Springer. 2020, pp. 171–182.
- [75] Mark D Wilkinson et al. "Addendum: The FAIR Guiding Principles for scientific data management and stewardship". In: *Scientific data* 6 (2019), p. 6.
- [76] Sandra Collins et al. *Turning FAIR into reality: Final report and action plan from the European Commission expert group on FAIR data*. 2018.
- [77] María Poveda-Villalón et al. "Coming to terms with FAIR ontologies". In: *International Conference on Knowledge Engineering and Knowledge Management*. Springer. 2020, pp. 255–270.
- [78] G Cota et al. "Best practices for implementing fair vocabularies and ontologies on the web". In: *Applications and Practices in Ontology Design, Extraction, and Reasoning* 49 (2020), p. 39.
- [79] Simon JD Cox et al. "Ten simple rules for making a vocabulary FAIR". In: *PLoS computational biology* 17.6 (2021), e1009041.
- [80] Stuart Weibel et al. "Dublin core metadata for resource discovery". In: *Internet Engineering Task Force RFC 2413.222* (1998), p. 132.
- [81] Dan Brickley and Libby Miller. "FOAF vocabulary specification 0.98". In: *Namespace document* 9 (2012).
- [82] Yves Raimond et al. "The Music Ontology." In: *ISMIR*. Vol. 2007. Citeseer. 2007, 8th.
- [83] Kevin Donnelly et al. "SNOMED-CT: The advanced terminology and coding system for eHealth". In: *Studies in health technology and informatics* 121 (2006), p. 279.
- [84] Libby Miller and Dan Brickley. *Schemarama*. URL: <https://danbri.org/words/2005/07/30/114>. 2005.
- [85] Libby Miller. "RDF Squish query language and Java implementation". In: *Institute for Learning and Research Technology, University of Bristol, disponible en ligne sur: <http://ilrt.org/discovery/2001/02/squish>* (2001).
- [86] Damian Steer and Libby Miller. "Validating RDF with TreeHugger and Schematron. Position Paper". In: *FOAF-Galway*. FOAF. 2004.
- [87] James Clark, Steve DeRose, et al. *XML path language (XPath)*. 1999.
- [88] Holger Knublauch. "SPIN-modeling vocabulary". In: *W3C Member Submission* 22 (2011).
- [89] Christian Fürber and Martin Hepp. "Using SPARQL and SPIN for data quality management on the semantic web". In: *International Conference on Business Information Systems*. Springer. 2010, pp. 35–46.
- [90] Shawn Simister and Dan Brickley. "Simple application-specific constraints for rdf models". In: *RDF Validation Workshop. Practical Assurances for Quality RDF Data, Cambridge, Ma, Boston*. 2013.
- [91] Dimitris Kontokostas et al. "Test-driven evaluation of linked data quality". In: *Proceedings of the 23rd international conference on World Wide Web*. 2014, pp. 747–758.

- [92] Jiao Tao et al. "Integrity constraints in OWL". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 24. 1. 2010, pp. 1443–1448.
- [93] Peter F Patel-Schneider. "Using description logics for RDF constraint checking and closed-world recognition". In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.
- [94] Kendall Clark and Evren Sirin. "On RDF validation, stardog ICV, and assorted remarks". In: *RDF Validation Workshop. Practical Assurances for Quality RDF Data, Cambridge, Ma, Boston*. 2013.
- [95] Arthur G Ryman, Arnaud Le Hors, and Steve Speicher. "OSLC Resource Shape: A language for defining constraints on Linked Data." In: *LDOW 996* (2013).
- [96] Karen Coyle and Tom Baker. "Dublin core application profiles. separating validation from semantics". In: *RDF Validation Workshop. Practical Assurances for Quality RDF Data, Cambridge, Ma, Boston*. 2013.
- [97] Peter M Fischer et al. "RDF constraint checking". In: (2015).
- [98] Eric Van der Vlist. *Relax ng: A simpler schema language for xml*. " O'Reilly Media, Inc.", 2003.
- [99] Eric Prud'hommeaux et al. "Shape Expressions Language 2.1". In: *W3C Community Group Report* (2019).
- [100] Eric Prud'hommeaux. "Shape Expressions (ShEx) JSON Formats". In: (2019).
- [101] Tim Bray. "The javascript object notation (json) data interchange format". In: (2014).
- [102] Eric Prud'hommeaux et al. "Development of a FHIR RDF data transformation and validation framework and its evaluation". In: *Journal of Biomedical Informatics* 117 (2021), p. 103755.
- [103] "The Gene Ontology resource: enriching a GOld mine". In: *Nucleic acids research* 49.D1 (2021), pp. D325–D334.
- [104] Guillermo Facundo Colunga et al. "ShEx-Lite: Automatic generation of domain object models from a shape expressions subset language?" In: *CEUR Workshop Proceedings*. 2020.
- [105] Herminio Garcia-Gonzalez and Jose Emilio Labra-Gayo. "XMLSchema2ShEx: Converting XML validation to RDF validation". In: *Semantic Web 11.2* (2020), pp. 235–253.
- [106] Gustavo Candela et al. "A shape expression approach for assessing the quality of linked open data in libraries". In: *Semantic Web Preprint* (2021), pp. 1–21.
- [107] Petri Leskinen, Eero Hyvönen, and Jouni Tuominen. "Members of Parliament in Finland Knowledge Graph and Its Linked Open Data Service". In: *Further with Knowledge Graphs*. IOS Press, 2021, pp. 255–269.
- [108] Holger Knublauch and Vladimir Alexiev. "SHACL Compact Syntax". In: *W3C Draft Community Group Report* (2018).
- [109] Holger Knublauch, Dean Allemang, and Simon Steyskal. "SHACL Advanced Features". In: *W3C Working Group Note* (2017).
- [110] Holger Knublauch, TopQuadrant, and Pano Maria. "SHACL JavaScript Extensions". In: *W3C Working Group Note* (2017).

- [111] Martin Leinberger et al. "Type checking program code using SHACL". In: *International Semantic Web Conference*. Springer. 2019, pp. 399–417.
- [112] Blerina Spahiu, Andrea Maurino, and Matteo Palmonari. "Towards Improving the Quality of Knowledge Graphs with Data-driven Ontology Patterns and SHACL." In: *ISWC (Best Workshop Papers)*. 2018, pp. 103–117.
- [113] Jesse Wright et al. "Schímatos: a SHACL-based web-form generator for knowledge graph editing". In: *International Semantic Web Conference*. Springer. 2020, pp. 65–80.
- [114] Sander Stolk and Kris McGlenn. "Validation of IfcOWL datasets using SHACL". In: *Proceedings of the 8th Linked Data in Architecture and Construction Workshop (LDAC2020) at Dublin, Ireland*. 2020.
- [115] Ranjith K Soman. "Modelling construction scheduling constraints using Shapes Constraint Language (SHACL)". In: *2019 European Conference on Computing in Construction, Chania, Greece*. 2019, pp. 351–358.
- [116] Umutcan Şimşek et al. "Domain-specific customization of schema.org based on SHACL". In: *International Semantic Web Conference*. Springer. 2020, pp. 585–600.
- [117] Andreas M Kaplan and Michael Haenlein. "Social media: back to the roots and back to the future". In: *Journal of systems and information technology* (2012).
- [118] Dan O'Sullivan. *Wikipedia: a new community of practice?* Routledge, 2016.
- [119] Denny Vrandečić. "Capturing meaning: Toward an abstract Wikipedia". In: (2018).
- [120] Fernanda B Viégas, Martin Wattenberg, and Kushal Dave. "Studying cooperation and conflict between authors with history flow visualizations". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. 2004, pp. 575–582.
- [121] Diana Maynard, Kalina Bontcheva, and Isabelle Augenstein. "Natural language processing for the semantic web". In: *Synthesis Lectures on the Semantic Web: Theory and Technology* 6.2 (2016), pp. 1–194.
- [122] Toms Bergmanis and Sharon Goldwater. "Context sensitive neural lemmatization with lemmatus". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. 2018, pp. 1391–1400.
- [123] Atefeh Farzindar and Diana Inkpen. "Natural language processing for social media". In: *Synthesis Lectures on Human Language Technologies* 8.2 (2015), pp. 1–166.
- [124] Joseph Weizenbaum. "ELIZA—a computer program for the study of natural language communication between man and machine". In: *Communications of the ACM* 9.1 (1966), pp. 36–45.
- [125] Pablo N Mendes et al. "DBpedia spotlight: shedding light on the web of documents". In: *Proceedings of the 7th international conference on semantic systems*. 2011, pp. 1–8.
- [126] Shijie Wu and Mark Dredze. "Beto, Bentz, Becas: The Surprising Cross-Lingual Effectiveness of BERT". In: *EMNLP*. 2019.
- [127] Przemysław Dymarski. *Hidden Markov models: Theory and applications*. BoD—Books on Demand, 2011.

- [128] Charles Sutton, Andrew McCallum, et al. "An introduction to conditional random fields". In: *Foundations and Trends® in Machine Learning* 4.4 (2012), pp. 267–373.
- [129] Jiuxiang Gu et al. "Recent advances in convolutional neural networks". In: *Pattern Recognition* 77 (2018), pp. 354–377.
- [130] Yong Yu et al. "A review of recurrent neural networks: LSTM cells and network architectures". In: *Neural computation* 31.7 (2019), pp. 1235–1270.
- [131] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).
- [132] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition)*. USA: Prentice-Hall, Inc., 2009. ISBN: 0131873210.
- [133] Yuri Lin et al. "Syntactic annotations for the google books ngram corpus". In: *Proceedings of the ACL 2012 system demonstrations*. 2012, pp. 169–174.
- [134] Mark Davies. "The Corpus of Contemporary American English as the first reliable monitor corpus of English". In: *Literary and linguistic computing* 25.4 (2010), pp. 447–464.
- [135] Bing Liu and Lei Zhang. "A survey of opinion mining and sentiment analysis". In: *Mining text data*. Springer, 2012, pp. 415–463.
- [136] Tommi Jauhiainen et al. "Automatic language identification in texts: A survey". In: *Journal of Artificial Intelligence Research* 65 (2019), pp. 675–782.
- [137] Charu C Aggarwal and ChengXiang Zhai. "A survey of text classification algorithms". In: *Mining text data*. Springer, 2012, pp. 163–222.
- [138] Balaji Krishnapuram et al. "Sparse multinomial logistic regression: Fast algorithms and generalization bounds". In: *IEEE transactions on pattern analysis and machine intelligence* 27.6 (2005), pp. 957–968.
- [139] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.
- [140] David Meyer and FT Wien. "Support vector machines". In: *The Interface to libsvm in package e1071* 28 (2015).
- [141] Abhishek Kumar, Jyotir Moy Chatterjee, and Vicente García Díaz. "A novel hybrid approach of SVM combined with NLP and probabilistic neural network for email phishing". In: *International Journal of Electrical and Computer Engineering* 10.1 (2020), p. 486.
- [142] C Jashubhai Rameshbhai and Joy Paulose. "Opinion mining on newspaper headlines using SVM and NLP". In: *International Journal of Electrical and Computer Engineering (IJECE)* 9.3 (2019), pp. 2152–2163.
- [143] Yuling Chen and Zhi Zhang. "Research on text sentiment analysis based on CNNs and SVM". In: *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*. IEEE. 2018, pp. 2731–2734.
- [144] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. ISSN: 1522-9602.
- [145] Mohammad-Parsa Hosseini et al. "Deep learning architectures". In: *Deep learning: concepts and architectures*. Springer, 2020, pp. 1–24.
- [146] Katherine Elkins and Jon Chun. "Can GPT-3 pass a writer's Turing Test?" In: *Journal of Cultural Analytics* 5.2 (2020), p. 17212.

- [147] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and Their Compositionality". In: NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, 3111–3119.
- [148] Jeffrey Pennington, Richard Socher, and Christopher D Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.
- [149] Diana Maynard, Kalina Bontcheva, and Dominic Rout. "Challenges in developing opinion mining tools for social media". In: *Proceedings of the@ NLP can u tag# usergeneratedcontent* (2012), pp. 15–22.
- [150] Amelia M Jamison, David A Broniatowski, and Sandra Crouse Quinn. "Malicious actors on Twitter: A guide for public health researchers". In: *American journal of public health* 109.5 (2019), pp. 688–692.
- [151] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).
- [152] Rolandos Alexandros Potamias, Georgios Siolas, and Andreas-Georgios Stafylopatis. "A transformer-based approach to irony and sarcasm detection". In: *Neural Computing and Applications* 32.23 (2020), pp. 17309–17320.
- [153] Palak Verma, Neha Shukla, and AP Shukla. "Techniques of sarcasm detection: A review". In: *2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*. IEEE. 2021, pp. 968–972.
- [154] Roberto González-Ibáñez, Smaranda Muresan, and Nina Wacholder. "Identifying sarcasm in twitter: a closer look". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. 2011, pp. 581–586.
- [155] Po-Yao Huang et al. "Multimodal filtering of social media for temporal monitoring and event analysis". In: *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*. 2018, pp. 450–457.
- [156] Donald Metzler, Congxing Cai, and Eduard Hovy. "Structured event retrieval over microblog archives". In: *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2012, pp. 646–655.
- [157] Jiawei Han, Micheline Kamber, and Jian Pei. "Data Mining: Concepts and Techniques Third Edition [M]". In: *The Morgan Kaufmann Series in Data Management Systems* 5.4 (2011), pp. 83–124.
- [158] Manoj Kumar Gupta and Pravin Chandra. "A comprehensive survey of data mining". In: *International Journal of Information Technology* 12.4 (2020), pp. 1243–1257.
- [159] Sara Radicati and J Levenstein. "Email Statistics Report, 2021-2025". In: *Radicati Group, Palo Alto, CA, USA, Tech. Rep* (2021).
- [160] Giang Nguyen et al. "Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey". In: *Artificial Intelligence Review* 52.1 (2019), pp. 77–124.
- [161] Ahmed Shihab Albahri et al. "Role of biological data mining and machine learning techniques in detecting and diagnosing the novel coronavirus (COVID-19): a systematic review". In: *Journal of medical systems* 44.7 (2020), pp. 1–11.

- [162] Xin-She Yang. *Introduction to algorithms for data mining and machine learning*. Academic press, 2019.
- [163] Han Wen et al. "Exploring user-generated content related to dining experiences of consumers with food allergies". In: *International Journal of Hospitality Management* 85 (2020), p. 102357.
- [164] Thien Hai Nguyen, Kiyooki Shirai, and Julien Velcin. "Sentiment analysis on social media for stock movement prediction". In: *Expert Systems with Applications* 42.24 (2015), pp. 9603–9611.
- [165] Archana Gupta et al. "Stock market prediction using data mining techniques". In: *2nd International Conference on Advances in Science & Technology (ICAST)*. 2019.
- [166] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. "A review on genetic algorithm: past, present, and future". In: *Multimedia Tools and Applications* 80.5 (2021), pp. 8091–8126.
- [167] Haoyuan Hong et al. "Applying genetic algorithms to set the optimal combination of forest fire related variables and model forest fire susceptibility based on data mining models. The case of Dayu County, China". In: *Science of the total environment* 630 (2018), pp. 1044–1056.
- [168] Ricardo JGB Campello et al. "Hierarchical density estimates for data clustering, visualization, and outlier detection". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 10.1 (2015), pp. 1–51.
- [169] YH Dovoedo and Subha Chakraborti. "Boxplot-based outlier detection for the location-scale family". In: *Communications in statistics-simulation and computation* 44.6 (2015), pp. 1492–1513.
- [170] Mingchen Feng et al. "Big data analytics and mining for effective visualization and trends forecasting of crime data". In: *IEEE Access* 7 (2019), pp. 106111–106123.
- [171] Ahmed Hussein Ali and Mahmood Zaki Abdullah. "Recent trends in distributed online stream processing platform for big data: Survey". In: *2018 1st Annual International Conference on Information and Sciences (AiCIS)*. IEEE. 2018, pp. 140–145.
- [172] Zainab Salih Ageed et al. "Comprehensive survey of big data mining approaches in cloud systems". In: *Qubahan Academic Journal* 1.2 (2021), pp. 29–38.
- [173] Albert Bifet et al. "Streamdm: Advanced data mining in spark streaming". In: *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. IEEE. 2015, pp. 1608–1611.
- [174] Francesco Corcoglioniti et al. "Processing billions of RDF triples on a single machine using streaming and sorting". In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. 2015, pp. 368–375.
- [175] Pritam Gundecha and Huan Liu. "Mining social media: a brief introduction". In: *New directions in informatics, optimization, logistics, and production* (2012), pp. 1–17.
- [176] Muhammad Aqib Javed et al. "Community detection in networks: A multidisciplinary review". In: *Journal of Network and Computer Applications* 108 (2018), pp. 87–111.

- [177] Xing Su et al. "A comprehensive survey on community detection with deep learning". In: *IEEE Transactions on Neural Networks and Learning Systems* (2022).
- [178] Huimin Huang et al. "Community-based influence maximization for viral marketing". In: *Applied Intelligence* 49.6 (2019), pp. 2137–2150.
- [179] Amir Moghaddam. "Detection of malicious user communities in data networks". PhD thesis. 2011.
- [180] Fei Tan, Yongxiang Xia, and Boyao Zhu. "Link prediction in complex networks: a mutual information perspective". In: *PloS one* 9.9 (2014), e107056.
- [181] Aron Culotta and Jennifer Cutler. "Mining brand perceptions from twitter social networks". In: *Marketing science* 35.3 (2016), pp. 343–362.
- [182] Muhammad Al-Qurishi et al. "User profiling for big social media data using standing ovation model". In: *Multimedia Tools and Applications* 77.9 (2018), pp. 11179–11201.
- [183] Rajesh Bose et al. "Analyzing political sentiment using Twitter data". In: *Information and communication technology for intelligent systems*. Springer, 2019, pp. 427–436.
- [184] Anitha Anandhan et al. "Social media recommender systems: review and open research issues". In: *IEEE Access* 6 (2018), pp. 15608–15628.
- [185] Allison JB Chaney, Brandon M Stewart, and Barbara E Engelhardt. "How algorithmic confounding in recommendation systems increases homogeneity and decreases utility". In: *Proceedings of the 12th ACM Conference on Recommender Systems*. 2018, pp. 224–232.
- [186] Ghayda Hassan et al. "Exposure to extremist online content could lead to violent radicalization: A systematic review of empirical evidence". In: *International journal of developmental science* 12.1-2 (2018), pp. 71–88.
- [187] Timothy La Fond and Jennifer Neville. "Randomization tests for distinguishing social influence and homophily effects". In: *Proceedings of the 19th international conference on World wide web*. 2010, pp. 601–610.
- [188] Luis V Casaló, Carlos Flavián, and Sergio Ibáñez-Sánchez. "Influencers on Instagram: Antecedents and consequences of opinion leadership". In: *Journal of business research* 117 (2020), pp. 510–519.
- [189] Felipe Bonow Soares, Raquel Recuero, and Gabriela Zago. "Influencers in polarized political networks on Twitter". In: *Proceedings of the 9th international conference on social media and society*. 2018, pp. 168–177.
- [190] Marijke De Veirman, Liselot Hudders, and Michelle R Nelson. "What is influencer marketing and how does it target children? A review and direction for future research". In: *Frontiers in psychology* 10 (2019), p. 2685.
- [191] Mariah L Wellman et al. "Ethics of authenticity: Social media influencers and the production of sponsored content". In: *Journal of Media Ethics* 35.2 (2020), pp. 68–82.
- [192] Xueting Liao et al. "Should We Trust Influencers on Social Networks? On Instagram Sponsored Post Analysis". In: *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE. 2021, pp. 1–8.
- [193] Xun Zhu, Youllee Kim, and Haseon Park. "Do messages spread widely also diffuse fast? Examining the effects of message characteristics on information diffusion". In: *Computers in human behavior* 103 (2020), pp. 37–47.

- [194] Xueqin Chen et al. "Modeling microscopic and macroscopic information diffusion for rumor detection". In: *International Journal of Intelligent Systems* 36.10 (2021), pp. 5449–5471.
- [195] K Nalini and L Jaba Sheela. "A survey on datamining in cyber bullying". In: *International Journal on Recent and Innovation Trends in Computing and Communication* 2.7 (2014), pp. 1865–1869.
- [196] Noriko Hara, Pnina Shachaf, and Khe Foon Hew. "Cross-cultural analysis of the Wikipedia community". In: *Journal of the American Society for Information Science and Technology* 61.10 (2010), pp. 2097–2108.
- [197] Sanmay Das, Allen Lavoie, and Malik Magdon-Ismael. "Manipulation among the arbiters of collective intelligence: How Wikipedia administrators mold public opinion". In: *ACM Transactions on the Web (TWEB)* 10.4 (2016), pp. 1–25.
- [198] Emily M Bender et al. "Annotating social acts: Authority claims and alignment moves in wikipedia talk pages". In: *Proceedings of the Workshop on Language in Social Media (LSM 2011)*. 2011, pp. 48–57.
- [199] Daniel Fernández-Álvarez et al. "Approaches to measure class importance in Knowledge Graphs". In: *PLOS ONE* 16.6 (June 2021), pp. 1–35.
- [200] Mark Craven, Johan Kumlien, et al. "Constructing biological knowledge bases by extracting information from text sources." In: *ISMB*. Vol. 1999. 1999, pp. 77–86.
- [201] Stephen P Borgatti and Martin G Everett. "A graph-theoretic perspective on centrality". In: *Social networks* 28.4 (2006), pp. 466–484.
- [202] Jon M Kleinberg. "Authoritative sources in a hyperlinked environment". In: *Journal of the ACM (JACM)* 46.5 (1999), pp. 604–632.
- [203] Heiko Paulheim and Christian Bizer. "Type inference on noisy rdf data". In: *International Semantic Web Conference*. Springer. 2013, pp. 510–525.
- [204] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. "Yago: a core of semantic knowledge". In: *Proceedings of the 16th international conference on World Wide Web*. ACM. 2007, pp. 697–706.
- [205] Douglas Foxvog. "Cyc". In: *Theory and Applications of Ontology: Computer Applications*. Springer, 2010, pp. 259–278.
- [206] George A Miller. *WordNet: An electronic lexical database*. MIT press, 1998.
- [207] Collin F Baker, Charles J Fillmore, and John B Lowe. "The berkeley framenet project". In: *COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics*. 1998.
- [208] Matteo Lissandrini et al. "Knowledge Graph Exploration: Where Are We and Where Are We Going?" In: *SIGWEB Newsl.* Summer 2020 (July 2020). ISSN: 1931-1745.
- [209] Dhavalkumar Thakker et al. "A note on intelligent exploration of semantic data". In: *Semantic Web* 10.3 (2019), pp. 525–527.
- [210] Šejla Čebirić et al. "Summarizing semantic graphs: a survey". In: *The VLDB Journal* 28.3 (2019), pp. 295–327.
- [211] Seyedamin Pouriyeh et al. "Ontology Summarization: Graph-Based Methods and Beyond". In: *International Journal of Semantic Computing* 13.02 (2019), pp. 259–283.

- [212] Paulo Orlando Queiroz-Sousa, Ana Carolina Salgado, and Carlos Eduardo Pires. "A method for building personalized ontology summaries". In: *Journal of Information and Data Management* 4.3 (2013), p. 236.
- [213] Georgia Troullinou et al. "RDFDigest+: A Summary-driven System for KBs Exploration." In: *International Semantic Web Conference (P&D/Industry/BlueSky)*. 2018.
- [214] Giannis Vassiliou et al. "Wbsum: workload-based summaries for RDF/S kbs". In: *33rd International Conference on Scientific and Statistical Database Management*. 2021, pp. 248–252.
- [215] Jimao Guo and Yi Wang. "Summarizing RDF graphs using Node Importance and Query History". In: *2021 International Conference on Service Science (ICSS)*. 2021, pp. 51–58.
- [216] William Webber, Alistair Moffat, and Justin Zobel. "A Similarity Measure for Indefinite Rankings". In: *ACM Trans. Inf. Syst.* 28.4 (Nov. 2010). ISSN: 1046-8188.
- [217] Pavel Berkhin. "A survey on pagerank computing". In: *Internet Mathematics* 2.1 (2005), pp. 73–120.
- [218] P Sargolzaei and F Soleymani. "Pagerank problem, survey and future research directions". In: *International Mathematical Forum*. Vol. 5. Citeseer. 2010, pp. 937–956.
- [219] Aric Hagberg, Pieter Swart, and Daniel S Chult. *Exploring network structure, dynamics, and function using NetworkX*. Tech. rep. Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.
- [220] Stanislav Malyshev et al. "Getting the most out of wikidata: Semantic technology usage in wikipedia's knowledge graph". In: *International Semantic Web Conference*. Springer. 2018, pp. 376–394.
- [221] Charles Spearman. "The proof and measurement of association between two things". In: *The American Journal of Psychology* 15 (1904), pp. 72–101.
- [222] David F Gleich. "PageRank beyond the Web". In: *SIAM Review* 57.3 (2015), pp. 321–363.
- [223] K Sparck Jones. "Automatic indexing". In: *Journal of documentation* 30.4 (1974), pp. 393–432.
- [224] Atish Das Sarma et al. "Fast distributed pagerank computation". In: *International Conference on Distributed Computing and Networking*. Springer. 2013, pp. 11–26.
- [225] Blerina Spahiu et al. "ABSTAT: ontology-driven linked data summaries with pattern minimalization". In: *European Semantic Web Conference*. Springer. 2016, pp. 381–395.
- [226] François Goasdoué, Pawel Guzewicz, and Ioana Manolescu. "RDF graph summarization for first-sight structure discovery". In: *The VLDB Journal* 2 (2020).
- [227] Ioana Manolescu. "Exploring RDF Graphs through Summarization and Analytic Query Discovery." In: *DOLAP*. 2020, pp. 1–5.
- [228] Carlos Eduardo Pires et al. "Summarizing ontology-based schemas in PDMS". In: *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*. IEEE. 2010, pp. 239–244.

- [229] Georgia Troullinou et al. "Exploring RDFS kbs using summaries". In: *International Semantic Web Conference*. Springer. 2018, pp. 268–284.
- [230] Michael Färber et al. "Linked data quality of dbpedia, freebase, opencyc, wikidata, and yago". In: *Semantic Web 9.1* (2018), pp. 77–129.
- [231] Andreas Thalhammer and Achim Rettinger. "PageRank on Wikipedia: Towards General Importance Scores for Entities". In: *The Semantic Web: ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 – June 2, 2016, Revised Selected Papers*. Cham: Springer International Publishing, Oct. 2016, pp. 227–240. ISBN: 978-3-319-47602-5.
- [232] Andreas Thalhammer and Achim Rettinger. "Browsing DBpedia entities with summaries". In: *European Semantic Web Conference*. Springer. 2014, pp. 511–515.
- [233] Andreas Thalhammer, Nelia Lasierra, and Achim Rettinger. "LinkSUM: using link analysis to summarize entity data". In: *International Conference on Web Engineering*. Springer. 2016, pp. 244–261.
- [234] Eun-kyung Kim and Key-Sun Choi. "Identifying global representative classes of DBpedia Ontology through multilingual analysis: A rank aggregation approach". In: *International Semantic Web Conference*. Springer. 2016, pp. 57–65.
- [235] Vooi Keong Boo and Patricia Anthony. "Agent for Mining of Significant Concepts in DBpedia". In: *Knowledge Technology*. Springer, 2012, pp. 313–322.
- [236] Giuseppe Pirrò. "Explaining and suggesting relatedness in knowledge graphs". In: *International Semantic Web Conference*. Springer. 2015, pp. 622–639.
- [237] Silvio Peroni, Enrico Motta, and Mathieu d'Aquin. "Identifying key concepts in an ontology, through the integration of cognitive principles with statistical and topological measures". In: *Asian Semantic Web Conference*. Springer. 2008, pp. 242–256.
- [238] Eleanor Rosch. "Principles of categorization". In: *Concepts: core readings* 189 (1999).
- [239] Michael Färber et al. "A Comparative Survey of DBpedia, Freebase, OpenCyc, Wikidata, and YAGO". In: *Semantic Web Journal*, July (2015).
- [240] Sungchan Park et al. "A Survey on Personalized PageRank Computation Algorithms". In: *IEEE Access* 7 (2019), pp. 163049–163062.
- [241] Sepandar Kamvar et al. "Exploiting the block structure of the web for computing pagerank". In: *Stanford University Technical Report* (2003).
- [242] Andrei Z Broder et al. "Efficient PageRank approximation via graph aggregation". In: *Information Retrieval* 9.2 (2006), pp. 123–138.
- [243] Antonio J Roa-Valverde and Miguel-Angel Sicilia. "A survey of approaches for ranking on the web of data". In: *Information Retrieval* 17.4 (2014), pp. 295–325.
- [244] Li Ding et al. "Swoogle: a search and metadata engine for the semantic web". In: *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. ACM. 2004, pp. 652–659.
- [245] Zaiqing Nie et al. "Object-level ranking: bringing order to web objects". In: *Proceedings of the 14th international conference on World Wide Web*. ACM. 2005, pp. 567–574.

- [246] Aidan Hogan, Stefan Decker, and Andreas Harth. "Reconrank: A scalable ranking method for semantic web data with context". In: *2nd Workshop on Scalable Semantic Web Knowledge Base Systems*. 2006.
- [247] Wang Wei, Payam Barnaghi, and Andrzej Bargiela. "Rational research model for ranking semantic entities". In: *Information Sciences* 181.13 (2011), pp. 2823–2840.
- [248] Roberto Mirizzi et al. "Ranking the linked data: the case of DBpedia". In: *International Conference on Web Engineering*. Springer. 2010, pp. 337–354.
- [249] Thomas Franz et al. "Triplerank: Ranking semantic web data by tensor decomposition". In: *International semantic web conference*. Springer. 2009, pp. 213–228.
- [250] Anila Sahar Butt, Armin Haller, and Lexing Xie. "DWRank: Learning concept ranking for ontology search". In: *Semantic Web 7.4* (2016), pp. 447–461.
- [251] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation". In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022.
- [252] Daniel Fernández-Álvarez, Jose Emilio Labra-Gayo, and Daniel Gayo-Avello. "Automatic extraction of shapes using sheXer". In: *Knowledge-Based Systems* (2021), p. 107975.
- [253] Daniel Fernández-Alvarez, Jose Emilio Labra-Gayo, and Herminio Garcia-González. *Inference and serialization of latent graph schemata using shex*. 2016.
- [254] Daniel Fernández-Álvarez et al. "Inference of Latent Shape Expressions Associated to DBpedia Ontology." In: *International Semantic Web Conference (P&D / Industry)*. 2018.
- [255] Jose Emilio Labra-Gayo et al. "Challenges in RDF validation". In: *Current Trends in Semantic Web Technologies: Theory and Practice*. Springer, 2019, pp. 121–151.
- [256] Iovka Boneva et al. "Shape designer for ShEx and SHACL constraints". In: *ISWC 2019-18th International Semantic Web Conference*. 2019.
- [257] Nandana Mihindukulasooriya et al. "RDF shape induction using knowledge base profiling". In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. 2018, pp. 1952–1959.
- [258] Harshvardhan J Pandit, Declan O'Sullivan, and Dave Lewis. "Using Ontology Design Patterns To Define SHACL Shapes." In: *WOP@ ISWC*. 2018, pp. 67–71.
- [259] Andrea Cimmino, Alba Fernández-Izquierdo, and Raúl García-Castro. "Astrea: automatic generation of SHACL shapes from ontologies". In: *European Semantic Web Conference*. Springer. 2020, pp. 497–513.
- [260] Pouya Ghasnezhad Omran et al. "Towards SHACL Learning from Knowledge Graphs." In: *ISWC (Demos/Industry)*. 2020, pp. 94–99.
- [261] Francisco Cifuentes-Silva, Daniel Fernández-Álvarez, and Jose Emilio Labra-Gayo. "National Budget as Linked Open Data: New Tools for Supporting the Sustainability of Public Finances". In: *Sustainability* 12.11 (2020), p. 4551.
- [262] Andra Waagmeester et al. "A protocol for adding knowledge to Wikidata: aligning resources on human coronaviruses". In: *BMC biology* 19.1 (2021), pp. 1–14.

- [263] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: a flexible data processing tool". In: *Communications of the ACM* 53.1 (2010), pp. 72–77.
- [264] Jose Emilio Labra Gayo, Dimitris Kontokostas, and Sören Auer. "Multilingual linked data patterns". In: *Semantic Web 6.4* (2015), pp. 319–337.
- [265] Renzo Arturo Alva Principe et al. "ABSTAT-HD: a scalable tool for profiling very large knowledge graphs". In: *The VLDB Journal* (2021), pp. 1–26.
- [266] Harshvardhan J Pandit, Declan O’Sullivan, and Dave Lewis. "An Argument for Generating SHACL Shapes from ODPs". In: *Advances in Pattern-Based Ontology Engineering*. IOS Press, 2021, pp. 134–141.
- [267] H Knublauch. "SHACL and OWL compared". In: URL: <https://spinrdf.org/shacl-and-owl.html> (2017).
- [268] Pouya Omran et al. *Active knowledge graph completion*. Tech. rep. Tech. rep., Australian National University, 2020. <https://openresearch...>, 2020.
- [269] Ji-Woong Choi. "Automatic Construction of SHACL Schemas for RDF Knowledge Graphs Generated by R2RML Mappings". In: *Journal of the Korea Society of Computer and Information* 25.8 (2020), pp. 9–21.
- [270] Thomas Delva et al. "RML2SHACL: RDF Generation Taking Shape". In: *Proceedings of the 11th on Knowledge Capture Conference*. 2021, pp. 153–160.
- [271] Anastasia Dimou et al. "RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data". In: *Proceedings of the 7th Workshop on Linked Data on the Web*. Ed. by Christian Bizer et al. Vol. 1184. CEUR Workshop Proceedings. Apr. 2014.
- [272] Adrien Basse et al. "DFS-based frequent graph pattern extraction to characterize the content of RDF Triple Stores". In: *Web Science Conference 2010 (WebSci10)*. 2010.
- [273] Eva Blomqvist et al. "Statistical Knowledge Patterns for Characterising Linked Data." In: *WOP*. Citeseer. 2013.
- [274] Jenny Rose Finkel, Trond Grenager, and Christopher D Manning. "Incorporating non-local information into information extraction systems by gibbs sampling". In: *Proceedings of the 43rd annual meeting of the association for computational linguistics (ACL’05)*. 2005, pp. 363–370.
- [275] Marius-Gabriel Butuc. "Semantically enriching content using openalais". In: *Editia* 9 (2009), pp. 77–88.
- [276] David Milne and Ian H Witten. "Learning to link with wikipedia". In: *Proceedings of the 17th ACM conference on Information and knowledge management*. 2008, pp. 509–518.
- [277] Andrea Moro, Alessandro Raganato, and Roberto Navigli. "Entity linking meets word sense disambiguation: a unified approach". In: *Transactions of the Association for Computational Linguistics* 2 (2014), pp. 231–244.
- [278] Ce Zhang et al. "Big data versus the crowd: Looking for relationships in all the right places". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2012, pp. 825–834.
- [279] Daojian Zeng et al. "Relation classification via convolutional deep neural network". In: *Proceedings of COLING 2014, the 25th international conference on computational linguistics: technical papers*. 2014, pp. 2335–2344.

- [280] Aldo Gangemi et al. "Semantic web machine reading with FRED". In: *Semantic Web 8.6* (2017), pp. 873–893.
- [281] Isabelle Augenstein, Sebastian Padó, and Sebastian Rudolph. "Lodifier: Generating linked data from unstructured text". In: *Extended Semantic Web Conference*. Springer. 2012, pp. 210–224.
- [282] Isabelle Augenstein, Diana Maynard, and Fabio Ciravegna. "Distantly supervised web relation extraction for knowledge base population". In: *Semantic Web 7.4* (2016), pp. 335–349.
- [283] Francesco Corcoglioniti, Marco Rospocher, and Alessio Palmero Aprosio. "Frame-based ontology population with PIKES". In: *IEEE Transactions on Knowledge and Data Engineering* 28.12 (2016), pp. 3261–3275.
- [284] Jose L Martinez-Rodriguez, Ivan Lopez-Arevalo, and Ana B Rios-Alvarado. "Mining information from sentences through Semantic Web data and Information Extraction tasks". In: *Journal of Information Science* (2020), p. 0165551520934387.
- [285] Nicolas Heist and Heiko Paulheim. "Language-Agnostic Relation Extraction from Wikipedia Abstracts". In: *The Semantic Web – ISWC 2017*. Ed. by Claudia d'Amato et al. Springer International Publishing, 2017, pp. 383–399.
- [286] Xin Dong et al. "Knowledge vault: A web-scale approach to probabilistic knowledge fusion". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 601–610.
- [287] William W Cohen. "Fast effective rule induction". In: *Machine learning proceedings 1995*. Elsevier, 1995, pp. 115–123.
- [288] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.
- [289] The Linguistic Society of Korea. "Linguistics in the Morning Calm". In: *Cognitive Linguistics Bibliography (CogBib)*. De Gruyter Mouton, 2010.
- [290] Clyde Holsapple. *Handbook on knowledge management 1: Knowledge matters*. Vol. 1. Springer Science & Business Media, 2013.
- [291] Hans Kamp. "A theory of truth and semantic representation". In: *Truth, interpretation and information* 277 (1984), p. 322.
- [292] Johan Bos. "Open-domain semantic parsing with boxer". In: *Proceedings of the 20th nordic conference of computational linguistics (NODALIDA 2015)*. 2015, pp. 301–304.
- [293] Christopher D Manning et al. "The Stanford CoreNLP natural language processing toolkit". In: *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 2014, pp. 55–60.
- [294] Paolo Ferragina and Ugo Scaiella. "Tagme: on-the-fly annotation of short text fragments (by wikipedia entities)". In: *Proceedings of the 19th ACM international conference on Information and knowledge management*. 2010, pp. 1625–1628.
- [295] Karin Kipper Schuler. *VerbNet: A broad-coverage, comprehensive verb lexicon*. University of Pennsylvania, 2005.
- [296] Roberto Navigli and Simone Paolo Ponzetto. "BabelNet: Building a very large multilingual semantic network". In: *Proceedings of the 48th annual meeting of the association for computational linguistics*. 2010, pp. 216–225.
- [297] Eneko Agirre and Aitor Soroa. "Personalizing pagerank for word sense disambiguation". In: *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*. 2009, pp. 33–41.

- [298] Silvio Peroni, Aldo Gangemi, and Fabio Vitali. "Dealing with markup semantics". In: *Proceedings of the 7th International Conference on Semantic Systems*. 2011, pp. 111–118.
- [299] Sebastian Hellmann et al. "Integrating NLP using linked data". In: *International semantic web conference*. Springer. 2013, pp. 98–113.
- [300] John Gruber. "Markdown: Syntax". In: URL <http://daringfireball.net/projects/markdown/syntax>. Retrieved on June 24 (2012), p. 640.
- [301] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual web search engine". In: *Computer networks and ISDN systems* 30.1 (1998), pp. 107–117.
- [302] Andrius Vabalas et al. "Machine learning algorithm validation with a limited sample size". In: *PloS one* 14.11 (2019), e0224365.
- [303] Denny Vrandečić. "Building a Multilingual Wikipedia". In: *Commun. ACM* 64.4 (2021), 38–41. ISSN: 0001-0782.
- [304] Ángel Obregón Sierra and Jorge Oceja Castanedo. "University students in the educational field and Wikipedia vandalism". In: *Proceedings of the 14th International Symposium on Open Collaboration*. 2018, pp. 1–7.
- [305] Juan R Martinez-Rico, Juan Martinez-Romo, and Lourdes Araujo. "Can deep learning techniques improve classification performance of vandalism detection in Wikipedia?" In: *Engineering Applications of Artificial Intelligence* 78 (2019), pp. 248–259.
- [306] Ammar Ismael Kadhim. "Survey on supervised machine learning techniques for automatic text classification". In: *Artificial Intelligence Review* 52.1 (2019), pp. 273–292.
- [307] David Šenkýř and Petr Kroha. "Patterns in textual requirements specification". In: *Proceedings of the 13th International Conference on Software Technologies*. 2018, pp. 197–204.
- [308] Esra A Abdelnabi, Abdelsalam M Maatuk, and Mohammed Hagal. "Generating UML Class Diagram from Natural Language Requirements: A Survey of Approaches and Techniques". In: *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering MI-STA*. IEEE. 2021, pp. 288–293.
- [309] Eman S Btoush and Mustafa M Hammad. "Generating ER diagrams from requirement specifications based on natural language processing". In: *International Journal of Database Theory and Application* 8.2 (2015), pp. 61–70.
- [310] Sutirtha Ghosh et al. "Automated generation of ER diagram from a given text in natural language". In: *2018 International Conference on Machine Learning and Data Engineering (iCMLDE)*. IEEE. 2018, pp. 91–96.
- [311] Marcin Michał Mirończuk. "Information Extraction System for Transforming Unstructured Text Data in Fire Reports into Structured Forms: A Polish Case Study". In: *Fire Technology* 56.2 (2020), pp. 545–581. ISSN: 1572-8099.
- [312] Chetan Arora et al. "Extracting Domain Models from Natural-Language Requirements: Approach and Industrial Evaluation". In: *MODELS '16*. Saint-malo, France: Association for Computing Machinery, 2016, 250–260. ISBN: 9781450343213.

- [313] Mihai Surdeanu et al. "Multi-instance multi-label learning for relation extraction". In: *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*. 2012, pp. 455–465.
- [314] Shingo Takamatsu, Issei Sato, and Hiroshi Nakagawa. "Reducing wrong labels in distant supervision for relation extraction". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2012, pp. 721–729.
- [315] Sebastian Riedel, Limin Yao, and Andrew McCallum. "Modeling relations and their mentions without labeled text". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2010, pp. 148–163.
- [316] André Freitas et al. "A Semantic Best-Effort Approach for Extracting Structured Discourse Graphs from Wikipedia." In: *WoLE@ ISWC 906 (2012)*, pp. 70–81.
- [317] Peter Exner and Pierre Nugues. "REFRACTIVE: An open source tool to extract knowledge from syntactic and semantic relations". In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. 2014, pp. 2584–2589.
- [318] Peter Exner and Pierre Nugues. "Entity Extraction: From Unstructured Text to DBpedia RDF triples." In: *WoLE@ ISWC*. 2012, pp. 58–69.
- [319] Daniel Gerber and Axel-Cyrille Ngonga Ngomo. "Extracting multilingual natural-language patterns for rdf predicates". In: *International Conference on Knowledge Engineering and Knowledge Management*. Springer. 2012, pp. 87–96.
- [320] Feng Niu et al. "DeepDive: Web-scale Knowledge-base Construction using Statistical Learning and Inference." In: *VLDS 12 (2012)*, pp. 25–28.
- [321] Ce Zhang. "DeepDive: a data management system for automatic knowledge base construction". PhD thesis. The University of Wisconsin-Madison, 2015.
- [322] Julia Hockenmaier and Mark Steedman. "CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank". In: *Computational Linguistics* 33.3 (2007), pp. 355–396.
- [323] Douglas Thain, Todd Tannenbaum, and Miron Livny. "Distributed computing in practice: the Condor experience". In: *Concurrency and computation: practice and experience* 17.2-4 (2005), pp. 323–356.
- [324] Dipanjan Das et al. "Frame-semantic parsing". In: *Computational linguistics* 40.1 (2014), pp. 9–56.
- [325] Anders Björkelund, Love Hafdell, and Pierre Nugues. "Multilingual semantic role labeling". In: *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*. 2009, pp. 43–48.
- [326] Martha Palmer, Daniel Gildea, and Paul Kingsbury. "The proposition bank: An annotated corpus of semantic roles". In: *Computational linguistics* 31.1 (2005), pp. 71–106.
- [327] Claire Bonial et al. "Propbank: Semantics of new predicate types". In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. 2014, pp. 3013–3019.

- [328] Adam Meyers et al. "The NomBank Project: An Interim Report". In: *HLT-NAACL 2004 Workshop: Frontiers in Corpus Annotation*. Ed. by A. Meyers. Boston, Massachusetts, USA: Association for Computational Linguistics, 2004, pp. 24–31.
- [329] Oren Etzioni et al. "Open information extraction from the web". In: *Communications of the ACM* 51.12 (2008), pp. 68–74.
- [330] Francesco Piccinno and Paolo Ferragina. "From TagME to WAT: a new entity annotator". In: *Proceedings of the first international workshop on Entity recognition & disambiguation*. 2014, pp. 55–62.
- [331] Jose L Martinez-Rodriguez, Ivan López-Arévalo, and Ana B Rios-Alvarado. "Openie-based approach for knowledge graph construction from text". In: *Expert Systems with Applications* 113 (2018), pp. 339–355.
- [332] Sebastian Krause et al. "Sar-graphs: A language resource connecting linguistic knowledge with semantic relations from knowledge graphs". In: *Journal of Web Semantics* 37 (2016), pp. 112–131.
- [333] Jose L Martinez-Rodriguez et al. "NLP and the Representation of Data on the Semantic Web". In: *Handbook of Research on Natural Language Processing and Smart Service Systems*. IGI Global, 2021, pp. 393–426.