



UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

MODELLING DYNAMICS OF RDF GRAPHS
WITH FORMAL CONCEPT ANALYSIS

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS
MENCIÓN COMPUTACIÓN

LARRY JAVIER GONZÁLEZ GONZÁLEZ

PROFESOR GUÍA:
AIDAN HOGAN

MIEMBROS DE LA COMISIÓN:
GONZALO NAVARRO BADINO
EDUARDO GODOY VEGA
CRISTIÁN RIVEROS JAEGER

SANTIAGO DE CHILE

2018

Resumen

La Web Semántica es una red de datos organizados de tal manera que permite su manipulación directa tanto por humanos como por computadoras. RDF es el *framework* recomendado por W3C para representar información en la Web Semántica. RDF usa un modelo de datos basado en grafos que no requiere ningún esquema fijo, provocando que los grafos RDF sean fáciles de extender e integrar, pero también difíciles de consultar, entender, explorar, resumir, etc.

En esta tesis, inspirados en *formal concept analysis* (un subcampo de las matemáticas aplicadas, basados en la formalización de conceptos y jerarquías conceptuales, llamadas *lattices*) proponemos un *data-driven schema* para grandes y heterogéneos grafos RDF. La idea principal es que si podemos definir un *formal context* a partir de un grafo RDF, entonces podemos extraer sus *formal concepts* y computar una *lattice* con ellos, lo que resulta en nuestra propuesta de esquema jerárquico para grafos RDF.

Luego, proponemos un álgebra sobre tales *lattices*, que permite (1) calcular deltas entre dos *lattices* (por ejemplo, para resumir los cambios de una versión de un grafo a otro), y (2) sumar un delta a un *lattice* (por ejemplo, para proyectar cambios futuros). Mientras esta estructura (y su álgebra asociada) puede tener varias aplicaciones, nos centramos en el caso de uso de modelar y predecir el comportamiento dinámico de los grafos RDF.

Evaluamos nuestros métodos al analizar cómo Wikidata ha cambiado durante 11 semanas.

Primero extraemos los conjuntos de propiedades asociadas a entidades individuales de una manera escalable usando el *framework* MapReduce. Estos conjuntos de propiedades (también conocidos como *characteristic sets*) son anotados con sus entidades asociadas, y posteriormente, con su cardinalidad. En segundo lugar, proponemos un algoritmo para construir la *lattice* sobre los *characteristic sets* basados en la relación de subconjunto. Evaluamos la eficiencia y la escalabilidad de ambos procedimientos.

Finalmente, usamos los métodos algebraicos para predecir cómo el esquema jerárquico de Wikidata evolucionaría. Contrastamos nuestros resultados con un modelo de regresión lineal como referencia. Nuestra propuesta supera al modelo lineal por un gran margen, llegando a obtener un *root mean square error* 12 veces más pequeño que el modelo de referencia.

Concluimos que, basados en *formal concept analysis*, podemos definir y generar un esquema jerárquico a partir de un grafo RDF y que podemos usar esos esquemas para predecir cómo evolucionarán, en un alto nivel, estos grafos RDF en el tiempo.

Abstract

The Semantic Web is a network of data organized in such a way that enables its use by both humans and computers. RDF is the framework recommended by the W3C to represent information on the Semantic Web. RDF uses a graph-based data model that does not need any fixed schema, making RDF graphs flexible to extend and integrate, but also more difficult to query, to understand, to explore, to summarize, etc.

In this thesis, we propose a novel data-driven schema for large-scale heterogeneous RDF graphs inspired by formal concept analysis, a subfield of applied mathematics based on the formalization of concepts and concept hierarchies, called lattices. The main idea is that if we can define a formal context from an RDF graph, then we can extract its formal concepts and compute a lattice with them, which results in our hierarchical data-driven schema proposal for RDF graphs.

While we argue that this lattice structure (and associated algebra) may have various applications, we focus on the use-case of modelling and predicting the dynamic behaviour of RDF graphs. We thus propose an algebra over such schema lattices, which allows us to compute a diff between two lattices (for example, to summarise the changes from one version of a graph to another), and to add diffs to lattices (for example, to project future changes).

We first extract the sets of properties associated with individual entities in a scalable way using the MapReduce framework. These property sets (aka. *characteristic sets*) are annotated with entity occurrences and later with cardinalities. Secondly, we propose an algorithm for constructing the lattice-based schema over the characteristic sets based on set-containment relations. We evaluate the efficiency and scalability of these procedures.

Finally, we use the algebraic methods to predict how the hierarchical schema of Wikidata would evolve. We contrast our results with a linear regression model as a baseline. Our proposal outperforms the linear model by a great margin. In the best case, when we predict the seventh week of Wikidata based on the six prior weeks, our method obtains a root mean squared error 12 times smaller and a mean absolute error 7 times smaller than the baseline.

We conclude that we can define and generate a hierarchical data-driven schema from diverse and large-scale RDF graphs based on formal concept analysis and that we can use those schemas to predict, in a high-level way, how the RDF graphs will evolve.

Agradecimientos

Quisiera agradecer muy especial y afectuosamente a Carme, Raisa y Roberto, mi familia, por su amor, cariño y apoyo incondicional. Los quiero.

Igualmente agradezco a mis amigos: Jocelyn, Rodrigo, Catalina, Patricio, Mauricio; quienes me han entregado nuevas ideas y puntos de vista y me han enseñado otra forma de ver el mundo, pero por sobre todo, por compartir y aumentar mis alegrías.

Agradezco sinceramente al cuerpo académico y administrativo de la Universidad de Chile, particularmente al DCC, por brindarme herramientas profesionales y académicas que hoy llevo conmigo. En especial agradezco a Cecilia Bastarrica, quien me orientó en tiempos de incertidumbre. A Aidan Hogan, mi profesor guía, por la disposición, ayuda, soporte y todo lo que me ha enseñado. Sin toda tu ayuda este trabajo no hubiera visto frutos. A Angélica Aguirre y Sandra Gaez, por su asistencia en cada trámite de este proceso. No puedo dejar fuera a todas las personas que, sin conocer sus nombres, ayudaron e hicieron más grata la permanencia en la U. A las tías de la biblioteca, facilitadores y auxiliares.

Table of contents

1	Introduction	1
1.1	Research Hypothesis	3
1.2	Objectives	3
1.2.1	General Objective	3
1.2.2	Specific Objectives	3
1.3	Contributions	4
1.4	Thesis Structure	4
1.5	Publications	5
2	Preliminaries	6
2.1	Lattice Theory	6
2.1.1	Sets	6
2.1.2	Tuples	7
2.1.3	Relations	8
2.1.4	Partially Ordered Sets	8
2.1.5	Lattices	9
2.2	Formal Concept Analysis	10
2.3	The Semantic Web	12
2.3.1	RDF	12
2.3.2	RDF Data Model	13
2.3.3	RDF Vocabularies	13
2.3.4	RDF Schema	15
2.4	Characteristic Sets	16
2.5	Wikidata	17
2.6	Apache Hadoop	19
2.6.1	Hadoop Distributed File System	19
2.6.2	Hadoop MapReduce	19
2.7	Linear Regression Models	20
2.8	Summary	21
3	Related Work	22
3.1	Computing Formal Concept Lattices	22
3.2	Formal Concept Analysis on the Semantic Web	23
3.3	Data-Driven RDF schema	24
3.4	Modelling dynamics on the Semantic Web	26
3.5	Summary	26

4	Proposed Graph Schema	27
4.1	RDF Concept Lattice	27
4.2	Characteristic Set Lattice	30
4.3	Characteristic Set #-Lattice	31
5	Modelling Dynamics in RDF Graphs	33
5.1	Characteristic Set Lattice Diff	33
5.2	Adding a Diff to a Lattice	35
6	Extraction of Characteristic Sets	38
6.1	Architecture	38
6.2	Data	38
6.3	MapReduce Jobs	39
6.4	Optimization and Performance Analysis	40
6.4.1	Number of Reducers	40
6.4.2	OIDs representation	41
6.4.3	Compression	42
6.4.4	Combiners	43
6.4.5	Summary	43
6.5	Characteristic Set Statistics	44
7	Characteristic Set Lattice Computation	46
8	High Level Dynamics Evaluation	49
8.1	Flat Prediction	50
8.2	Transitive Prediction	50
9	Conclusions	52
	Bibliography	53

Chapter 1

Introduction

Several years have passed since the winner of the 2016 Turing Award, Sir Timothy John Berners-Lee, along with other researchers, created what we know today as the Web. It seems distant the time where there wasn't a uniform way to link and retrieve documents. On the contrary, today it is almost natural to start your personal computer, open your default or preferred browser and then visit your favorite sites looking for news, information or just only fun.

The Web solved the problem to connect documents – or web pages – over a global set of heterogeneous computers. Now users of the Web can seamlessly follow links from documents stored on machines in the United States to documents stored in Singapore or South Africa. Furthermore, with knowing the exact name and location of a document, the Web allows users to use search engines to look for information.

The Web has been a place where several organizations have been born. Among them, *Wikipedia*¹, a free online encyclopedia that anyone around the world can edit [9], is one of the top five websites on the Web². Wikipedia allows users – called collaborators – to create, edit and publish documents. Those documents can then be read by anyone on the Web. Wikipedia was created in 2001. Since then, more than 47 million articles have been published in more than 290 languages³.

As an example, when a user wants to know the protagonist's name of his favorite movie, he could access Wikipedia, search for the movie's title and then look for the cast. But, could the same user just directly ask – instead of searching documents – for the actor's name?

Berners-Lee not only participated in the creation of the Web but also articulated another vision: the Semantic Web. While the Web is a network of documents designed to be read by humans, the Semantic Web is a network of data designed to be read and manipulated by both humans and computers [12]. In Berners Lee's dream, an *Agent* – specialized software – could perform tasks, retrieve information, set up appointments, understand data, conduct

¹<https://www.wikipedia.org/>

²<https://www.alexa.com/topsites>. Accessed on 28.01.2018.

³https://meta.wikimedia.org/wiki/List_of_Wikipedias. Accessed on 28.01.2018.

automated reasoning, etc.

Following our previous example, on the Semantic Web, a user could ask for the name of the protagonist of his favorite movie without looking for the movie. The same user could then directly ask for other movies involving that actor and director, or find the most successful actor in the movie based on earnings. The integration of data from multiple sources would be performed automatically.

The Semantic Web uses graph databases, specifically based on RDF [22] to store information. As opposed to relational databases, where fixing the schema is one of the first tasks, graph databases do not need to set any schema, making its flexibility one of its principal characteristics.

However, each relational database engine has commands to list all tables and to display the details of a specific one, i.e. it is an easy task to see the schema and a summary of the data. Besides, when a new record is added into a relational database, it needs to fit in the fixed schema and satisfy the conditions of data completeness (recorded in the same schema). This does not occur in graph databases, where there is typically no fixed schema nor data restrictions. Because of this flexibility, some problems arise: graph databases are difficult to query, to understand, to explore and to summarize. It is not easy to see what a graph database contains, how it will evolve or if its data is complete.

In recent years several works have been done related to this issue. Statistics of RDF data have been used to improve SPARQL⁴ – the query language for RDF databases – performance [59]; RDF graph summaries have been used to assist SPARQL query formulation [17] and to help RDF application designers [18]; relational schemas have been derived from RDF datasets to improve the efficiency of RDF databases [54]; RDF profilers have been developed to discover synonyms among predicates and to suggest missing facts [4]; and so forth. Regardless of the application, all these authors have implicitly proposed that computing a data-driven schema from an RDF graph is useful.

In this work, we propose another schema for RDF graphs inspired by *formal concept analysis* (FCA), a subfield of applied mathematics based on the formalization of concepts, and concept hierarchies called lattices [62]. Given a large and diverse RDF dataset, we develop a scalable way to extract its *characteristic sets* [50], our approximation to concepts, and then we build a schema – or lattice – over them. Furthermore, we propose an algebra over such lattices, which allows us (a) to subtract two lattices and generate a *delta* – or difference – as a measure of changes, and (b) to add a delta to a lattice.

Afterwards, we apply our schema proposal by modeling dynamics in RDF graphs. To do so, we extracted the characteristic sets and generated lattices for 11 versions of *Wikidata*⁵, the semantic sister of Wikipedia, which provides large and heterogeneous dumps of RDF data on a weekly basis. We then apply our lattice algebra to predict how the schema of Wikidata would evolve. Finally, we contrast our predictions with the results of a baseline linear regression model.

⁴<https://www.w3.org/TR/rdf-sparql-query/>

⁵<https://www.wikidata.org/>

All data, source code and other evaluation materials are available at:
https://github.com/larryjgonzalez/rdf_dynamics.

1.1 Research Hypothesis

While several works have been conducted to generate schemas of graph databases and to study the dynamics of networks in an entity level, to the best of our knowledge, there hasn't been any study that addresses the dynamics of diverse and large graph databases in a high-level approach. Consequently, we combine previous ideas to establish our research hypothesis:

From diverse, large-scale graph datasets like Wikidata, it is feasible to compute a hierarchical schema based on formal concept analysis where characteristic sets are considered an approximation to formal concepts. Furthermore, such a notion of schema can be used to model and predict high-level changes in such dataset better than a baseline linear regression model.

While the hierarchical schema we propose is useful for a variety of applications, including query processing, visualization, assessing data completeness, data summaries, etc. we currently focus on investigating and evaluating the use-case of modelling and predicting changes in RDF graphs.

We select Wikidata as an experimental dataset since it contains billions of triples and is edited by thousands of collaborators⁶, making it a highly diverse and dynamic data source. On the other hand, it provides weekly dumps of data that contain inherent dynamic behavior. Hence we see Wikidata as a very challenging and important target dataset over which to apply our proposed methods. It is out of the current scope to evaluate our methods for other use-cases and datasets.

1.2 Objectives

1.2.1 General Objective

The primary aim of this thesis is to propose a novel form of schema for RDF graphs and to evaluate its effectiveness for predicting future changes in such graphs.

1.2.2 Specific Objectives

Towards investigating the aforementioned research hypothesis, we identify the following specific objectives:

⁶www.wikidata.org/wiki/Wikidata:Statistics

- to define a new notion of hierarchical schema for RDF graphs.
- to propose algorithms to compute such a notion of hierarchical schema over diverse and large-scale RDF graphs.
- to define a subtraction operator over schemas that generate a diff (or "delta") summarizing changes between two versions of the dataset.
- to define an addition operator between a schema and a diff to predict future high-level changes to the dataset.
- to apply and evaluate our methods for predicting changes in the RDF graph of Wikidata.

1.3 Contributions

- We propose a notion of hierarchical schema for RDF graphs inspired by formal concept analysis.
- We present a scalable way of computing characteristics sets from a large and heterogeneous dataset.
- We present an algorithm for computing characteristics sets lattices, which result in our hierarchical schema proposal.
- We propose an algebra for computing both: a high-level diff between schemas and a high-level addition between a schema and a delta.
- We evaluate our method by extracting lattices for 11 weekly versions of the Wikidata RDF graph. We also present performance and scalability analysis.
- We show the utility of our hierarchical schema for predicting how Wikidata will evolve, and we contrast our predictions with the results of a standard prediction technique: simple linear regression.

1.4 Thesis Structure

This thesis is organized as follows.

- In Chapter 2, we provide preliminaries on lattice theory, formal concept analysis, the Semantic Web, characteristic sets, Wikidata, Apache Hadoop, and linear regression models.

- In Chapter 3, we discuss related works in four areas pertaining to this work. First, we review algorithm to extract formal concepts and build lattices. Second, we review how formal concept analysis, as a research field, has been applied to the Semantic Web. Third, we review different approaches to generate data-driven schemas from RDF graphs. And finally, we review other efforts to model dynamics on the Semantic Web.
- In Chapter 4, we present our schema proposal for RDF graphs.
- In Chapter 5, we define an algebra over our schema proposal, which allows us to subtract two schemas and add the resulting diff to another schema.
- In Chapter 6, we present a scalable implementation of our schema proposal, and we present a performance analysis over Wikidata.
- In Chapter 7, we present an algorithm to compute lattices over our characteristic sets and present the results for Wikidata.
- In Chapter 8, we present an evaluation of our method by extracting lattices for 11 versions of Wikidata. We also compare our predictions of future schemas with a baseline linear regression method, predicting the changes of Wikidata over 11 weeks in a high-level manner.
- Finally, in Chapter 9, we present the conclusions of this thesis, and we discuss some future work.

1.5 Publications

The main results of this thesis have been accepted for publication in the following international conference and workshop:

- Larry González and Aidan Hogan. 2018. “Modelling Dynamics in Semantic Web Knowledge Graphs with Formal Concept Analysis”. In the Proceedings of WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France. ACM, New York, NY, USA 10 Pages. <https://doi.org/10.1145/3178876.3186016> [37].
- Larry González and Aidan Hogan. 2018. “A Data-Driven Graph Schema.” In the Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Cali, Colombia, May 23-25, 2018 [36].

Chapter 2

Preliminaries

In this chapter, we provide an introduction to several and diverse topics on which we base this thesis. We start with the mathematical background, where we review lattice theory and formal concept analysis. We then visit the Semantic Web, focusing on the Resource Description Framework (RDF), characteristic sets, and Wikidata – the semantic sister of Wikipedia. We also summarize a framework for distributed computing called Hadoop, and finally, we review the classical linear regression model.

2.1 Lattice Theory

In this section, we summarize some basic mathematical concepts that are going to be used later in this thesis. We start from elemental concepts, covering sets, tuples, and relations. We then briefly cover partially ordered sets and lattices.

2.1.1 Sets

A set – or collection – is a mathematical structure used to group things. For example, the collection of letters a , b , and c is a set. It is represented by $S = \{a, b, c\}$, where S is the name of the set. We use the notation $a \in S$ to equivalently express: (1) a is an element of S , (2) a is in S , and (3) S contains a . On the contrary, we use $d \notin S$ represents that d is not an element of S .

Sets can be represented by (1) listing all its elements delimited by brackets (as we previously defined S), or (2) specifying properties that its elements need to satisfy. As an example, the set of multiples of three is $\{x \in \mathbb{N} : \text{mod}(x, 3) = 0\}$. A set of zero elements is called the empty set and it is denoted by \emptyset . A set of one element is called a singleton.

There are three principal operations between sets: *union*, *intersection*, and *difference*. The union (\cup) of sets S_1 and S_2 is a new set containing elements of S_1 and S_2 , i.e. $S_1 \cup S_2 =$

$\{x : x \in S_1 \text{ or } x \in S_2\}$. Analogously, the intersection (\cap) of S_1 and S_2 is the set containing elements present in S_1 and S_2 . i.e. $S_1 \cap S_2 = \{x : x \in S_1 \text{ and } x \in S_2\}$. Finally, the difference ($/$) of S_1 and S_2 is the set containing elements of S_1 less the elements of S_2 , i.e. $S_1/S_2 = \{x : x \in S_1 \text{ and } x \notin S_2\}$.

A set S_1 is a *subset* of S_2 ($S_1 \subseteq S_2$) if and only if $S_1 \cap S_2 = S_1$, i.e., all elements of S_1 are also in S_2 . Two sets, S_1 and S_2 are equal if and only if $S_1 \subseteq S_2$ and $S_2 \subseteq S_1$, in other words, they have the same elements. S_1 and S_2 are *disjoint* if $S_1 \cap S_2 = \emptyset$, i.e., they do not share any element.

Let S be a set. A *partition* of S is a set of non-empty sets $\{S_1, \dots, S_n\}$ such that they are disjoint between them (if $i \neq j$, then $S_i \cap S_j = \emptyset$) and its union form S , i.e., $\bigcup_{1 \leq i \leq n} S_i = S$. In other words, each element of S appears in exactly one S_i . On the other hand, the *power set* of S , represented by 2^S , is the set of all subsets of S . As an example, let $S = \{a, b, c\}$, then a partition of S could be $\{\{a\}, \{b, c\}\}$, while $2^S = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$.

Sets do not distinguish repetition nor order, i.e. $\{a, b, c\} = \{c, a, b\} = \{b, b, c, a\}$. Nevertheless, being able to order elements is a property useful for many applications. In the next section, we cover *tuples*: another algebraic structure which deals with order.

2.1.2 Tuples

Definition 2.1 (Ordered Pair). *An ordered pair is a set of two elements where the order is essential. To delimit an ordered pair, we use parentheses instead of brackets.*

Example 2.1. *Let a and b be two objects. There are four possible ordered pairs: (a, a) , (a, b) , (b, a) , and (b, b) . Note that $(a, b) \neq (b, a)$ because order matters. Also, there is no restriction about a appearing twice.*

Definition 2.2 (Tuple). *Let a_1, \dots, a_n be n elements. The ordered set (a_1, \dots, a_n) is called an n -tuple, or simply, a tuple. When a tuple contains three elements, it is called a triple.*

Example 2.2. *(s, p, o) , is a 3-tuple (or triple) where s is the first element, p is the second element, and o is the third element.*

Definition 2.3 (Cartesian Product). *Let A, B be two non-empty sets. The cartesian product of A and B , represented by $A \times B$ is the set of all ordered pairs (a, b) such that $a \in A$ and $b \in B$, i.e. $A \times B = \{(a, b) : a \in A \text{ and } b \in B\}$.*

Now we can group things using sets and distinguish when an element appears before another using tuples. In the next section, we cover how to create relations between the elements of two (or more) sets.

2.1.3 Relations

Definition 2.4 (Relation). Let A and B be two non-empty sets. A binary relation \mathcal{R} between A and B is a subset of $A \times B$. We use the notation $a\mathcal{R}b$ to say that $(a, b) \in \mathcal{R}$. When $A = B$, we say that \mathcal{R} is a binary relation on A . A binary relation \mathcal{R} on A can be:

- Reflexive $\Leftrightarrow \forall a \in A, a\mathcal{R}a$
- Symmetric $\Leftrightarrow \forall (a, b) \in A, a\mathcal{R}b \Rightarrow b\mathcal{R}a$
- Antisymmetric $\Leftrightarrow \forall (a, b) \in A, a\mathcal{R}b \wedge b\mathcal{R}a \Rightarrow a = b$
- Transitive $\Leftrightarrow \forall (a, b, c) \in A, a\mathcal{R}b \wedge b\mathcal{R}c \Rightarrow a\mathcal{R}c$

Example 2.3. Let us consider the binary relation \mathcal{R} in \mathbb{N} , defined by $x\mathcal{R}y := \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid x \leq y\}$. \mathcal{R} is reflexive because $\forall x \in \mathbb{N}, x \leq x$. It is not symmetric: just consider $x = 2$ and $y = 4$, then $x\mathcal{R}y$ is true, but $y\mathcal{R}x$ is false. On the contrary, \mathcal{R} is antisymmetric because $x\mathcal{R}y$ and $y\mathcal{R}x$ implies that $x = y$. Finally, it is direct to see that \mathcal{R} is transitive.

Definition 2.5 (Equivalence Relation). A relation \mathcal{R} is said to be an equivalence relation if it is reflexive, symmetric and transitive.

Definition 2.6 (Partial Order Relation). A relation \mathcal{R} is said to be a partial order relation if it is reflexive, antisymmetric and transitive. A partial order relation is often denoted by the symbol \leq .

2.1.4 Partially Ordered Sets

Definition 2.7 (Partially Ordered Set). A partially ordered set, or simply poset, is a pair (A, \leq) where A is a non-empty set and \leq is a partial order relation on A .

Example 2.4. Let A be the set of letters in English alphabet. The pair $(2^A, \subseteq)$, where 2^A is the power set of A and \subseteq is the subset relation, is a partially ordered set.

In effect, by construction 2^A is a non-empty set. On the other hand, \subseteq is a partial order relation on A because: (1) It is reflexive: any set is a subset of itself. (2) It is antisymmetric: let X, Y be sets such that $X \subseteq Y$ and $Y \subseteq X$; then any element in X is also in Y and any element in Y is also in X which means $X = Y$. (3) It is transitive: let X, Y, Z be sets such that $X \subseteq Y$ and $Y \subseteq Z$. Then, all elements in X are in Y , and all elements of Y are in Z , which means that all elements of X are also in Z .

Definition 2.8 (Infimum and Supremum). *Let (A, \leq) be a partially ordered set, $X \subseteq A$ such that $X \neq \emptyset$, and $\alpha \in A$. Then:*

- α is a lower bound of X if $(\forall x \in X) \alpha \leq x$.
- α is an upper bound of X if $(\forall x \in X) x \leq \alpha$.
- α is the infimum of X , denoted by $\inf(X)$, if (1) α is a lower bound of X , and (2) $(\forall x \in A \mid x \text{ is a lower bound of } X) x \leq \alpha$.
- α is the supremum of X , denoted by $\sup(X)$, if (1) α is an upper bound of X , and (2) $(\forall x \in A \mid x \text{ is an upper bound of } X) \alpha \leq x$.

Note that a set may have zero, one or more lower and upper bounds, while it may have zero or only one infimum and supremum. The infimum, denoted by $\inf(\cdot)$, is also called the *greatest lower bound*. On the other hand, the supremum, denoted by $\sup(\cdot)$, is also called the *least upper bound*.

Example 2.5. Let A be the set of letters in English alphabet, $(2^A, \subseteq)$ a partially ordered set, and $X = \{\{a, b, c\}, \{a, b, d\}\}$, where a, b, c , and d are actual letters. The sets in $2^{\{a, b\}}$ are lower bounds of X . On the other hand, all sets in $\{Y \in 2^A \mid \{a, b, c, d\} \subseteq Y\}$ are upper bounds of X . The infimum of X is $\inf(X) = \{a, b\}$, while its supremum is $\sup(X) = \{a, b, c, d\}$.

2.1.5 Lattices

Definition 2.9 (Lattice). *Let $\mathcal{L} = (A, \leq)$ be a partially ordered set. \mathcal{L} is called a lattice if $\inf(\{a, b\})$ and $\sup(\{a, b\})$ exist $\forall a, b \in A$. They are usually represented in a Hasse diagram, where the nodes are the elements of A , and the edges represent that two elements are related by \leq .*

Example 2.6. Let $A = \{a, b, c\}$ be a set, $(2^A, \subseteq)$ be a poset and $X = \{x \in 2^A \mid a \in x\}$. Figure 2.1 represents the Hasse diagram of this poset.

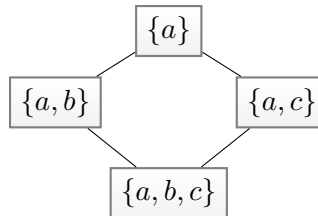


Figure 2.1: Hasse diagram of Example 2.6. The nodes of this diagram are in $\{x \in 2^{\{a, b, c\}} \mid a \in x\}$ and the edges represent the direct subset relations.

2.2 Formal Concept Analysis

Formal concept analysis is a subfield of applied mathematics based on the formalization of concepts, and concept hierarchies [62]. Essentially, it is a mathematical model to speak about entities (a.k.a. objects), attributes (a.k.a. properties) and the relations between them, i.e., to model that an entity has an attribute.

Definition 2.10 (Formal Context). *A formal context is a triple $X = (E, A, I)$, where E is a set of entities, A a set of attributes, and I a binary relation between E and A . We say that an entity $e \in E$ has an attribute $a \in A$ if and only if $(e, a) \in I$, or equivalently, eIa (see Definition 2.4).*

Towards defining formal concepts, we need to provide two operators. Given $X = (E, A, I)$ a formal context and $E_1 \subseteq E$ a set of entities, let $\langle\langle E_1 \rangle\rangle_X := \{a \in A \mid \forall e \in E_1 : (e, a) \in I\}$ be the set of attributes that a set of entities share; conversely, for a set of attributes $A_1 \subseteq A$, let $\langle\langle A_1 \rangle\rangle_X := \{e \in E \mid \forall a \in A_1 : (e, a) \in I\}$ be the set of entities that a set of attributes share.

Remark 2.1. *Let $X = (E, A, I)$ be a formal context, $E_1, E_2 \subseteq E$, and $A_1, A_2 \subseteq A$. Then the $\langle\langle \cdot \rangle\rangle_X$ operator satisfies:*

- $(\forall E_1, E_2 \subseteq E), E_1 \subseteq E_2 \Rightarrow \langle\langle E_2 \rangle\rangle_X \subseteq \langle\langle E_1 \rangle\rangle_X$
- $(\forall A_1, A_2 \subseteq A), A_1 \subseteq A_2 \Rightarrow \langle\langle A_2 \rangle\rangle_X \subseteq \langle\langle A_1 \rangle\rangle_X$
- $(\forall E_1 \subseteq E), E_1 \subseteq \langle\langle\langle\langle E_1 \rangle\rangle_X\rangle\rangle_X$ and $\langle\langle E_1 \rangle\rangle_X = \langle\langle\langle\langle\langle\langle E_1 \rangle\rangle_X\rangle\rangle_X\rangle\rangle_X$
- $(\forall A_1 \subseteq A), A_1 \subseteq \langle\langle\langle\langle A_1 \rangle\rangle_X\rangle\rangle_X$ and $\langle\langle A_1 \rangle\rangle_X = \langle\langle\langle\langle\langle\langle A_1 \rangle\rangle_X\rangle\rangle_X\rangle\rangle_X$

Definition 2.11 (Formal Concept). *Let $X = (E, A, I)$ be a formal context, $E_1 \subseteq E$, and $A_1 \subseteq A$. A formal concept of X is an ordered pair (E_1, A_1) where $\langle\langle E_1 \rangle\rangle = A_1$, and $\langle\langle A_1 \rangle\rangle = E_1$. A_1 and E_1 are called the extent and the intent of the formal concept respectively. We use \mathbb{C} to denote the set of all formal concepts of X .*

Definition 2.12 (Subconcept - Superconcept Relation). *Let $X = (E, A, I)$ be a formal context, \mathbb{C} the set of all formal concepts of X , and $\mathbb{C}_1 = (E_1, A_1)$, $\mathbb{C}_2 = (E_2, A_2)$ two formal concepts in \mathbb{C} . We say that \mathbb{C}_1 is a subconcept of \mathbb{C}_2 , or that \mathbb{C}_2 is a superconcept of \mathbb{C}_1 , both denoted by $\mathbb{C}_1 \leq \mathbb{C}_2$, if and only if $E_1 \subseteq E_2$. Note that due to Remark 2.1, this is equivalent to $\mathbb{C}_1 \leq \mathbb{C}_2 \Leftrightarrow A_2 \subseteq A_1$.*

Remark 2.2 (Concept Lattice). *Let \mathbb{C} denote the set of all formal concepts of X , then (\mathbb{C}, \leq) is not only a partially ordered set (per Definition 2.7), but also a lattice (per Definition 2.9) called the concept lattice.*

Example 2.7. *Let us consider some people related to the movie industry. When Robert De Niro was young, he starred in several movies like Taxi Driver, The Deer Hunter, and Raging Bull. As his career progressed, Robert also produced some movies, like Thunderheart and The Irishman while he only directed two: A Bronx Tale and The Good Shepherd. But, he never*

stopped being an actor, as a matter of fact, he starred in Jackie Brown too. In this industry, some people maintain focus in acting, like Christopher Walken (starred in The Deer Hunter), while others develop different roles, like Julianne Moore, who has been actress and producer at the same time (Marie and Bruce) or Samuel L. Jackson, who co-starred with Robert in Jackie Brown and produced Eve's Bayou later. Jackie Brown was directed by Quentin Tarantino, who is the producer of Death Proof and Hostel. To make our example more interesting, let us consider Sebastian Lelio, the director of A Fantastic Woman, where Daniela Vega was the protagonist. This movie was produced by Pablo Larrain, who is the producer of Gloria 2, where Julianne Moore is going to be the star.

Figure 2.7 represents the information about people and roles in the previous paragraph. Towards defining a formal context X , we just need to identify E as the set of people, A as the set of roles, and I as the matrix components with ticks in it. As a consequence, this matrix (called a cross table in literature) represents a formal context.

Name	star	direct	produce
Robert De Niro	✓	✓	✓
Christopher Walken	✓		
Julianne Moore	✓		✓
Samuel L. Jackson	✓		✓
Quentin Tarantino		✓	✓
Sebastian Lelio		✓	
Daniela Vega	✓		
Pablo Larrain			✓

Figure 2.2: Examples of movie stars, directors and producers.

In the following we are going to reference the people's names and roles only with their initials; thus, RN represents Robert De Niro, and s represents star. We use upper cases in names, and lower cases in roles (properties).

Let us note that $\langle\langle\{SJ\}\rangle\rangle_X$, i.e., the set of properties that the set $\{SJ\}$ has, is equal to $\{s, p\}$. But, when we compute $\langle\langle\{s, p\}\rangle\rangle_X$, we obtain $\{RN, JM, SJ\}$. Now, let us consider $\langle\langle\{RN, JM, SJ\}\rangle\rangle_X = \{s, p\}$. Over these three steps we have obtained one formal concept: $(\{RN, JM, SJ\}, \{s, p\})$, because $\langle\langle\{RN, JM, SJ\}\rangle\rangle_X = \{s, p\}$ and $\langle\langle\{s, p\}\rangle\rangle_X = \{RN, JM, SJ\}$. Note that we also satisfy Remark 2.1.

Figure 2.3 shows the formal concepts from the running example organized in a lattice according to the subconcept relation (See Definition 2.12).

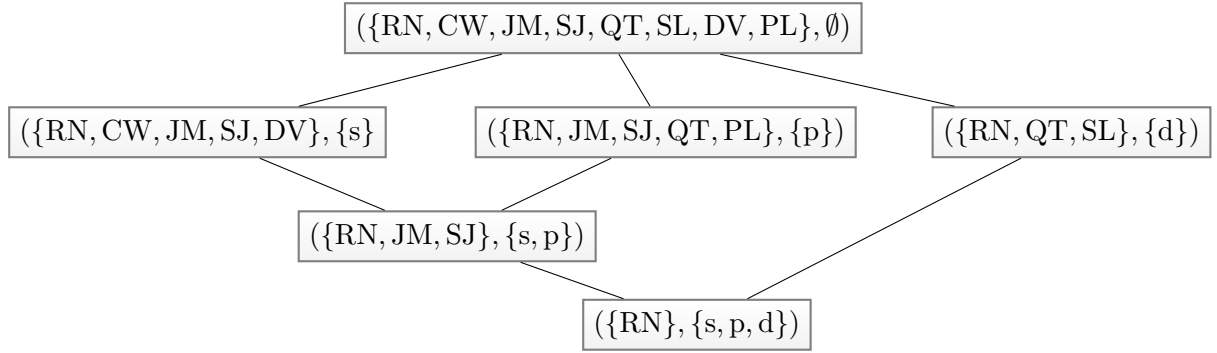


Figure 2.3: Concept lattice – represented by a Hasse Diagram – of people from the movie industry and their roles. Each node is a formal concept, and the edges are the subconcept relation.

The data-driven schema that we propose for RDF graphs is inspired by formal concept analysis, which will be described in Chapter 4. Next we provide some general background on the Semantic Web, for which RDF is the core data model.

2.3 The Semantic Web

The Semantic Web is a web of data. While the Web is a network of interconnected documents designed to be read by humans, the Semantic Web is a network of interconnected data designed to be read and manipulated by both humans and computers. In the words of Tim Berners Lee, *The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation*[12].

To achieve this, on the Semantic Web, data is structured in a meaningful way that allows software agents to retrieve, analyze, interpret, manipulate and communicate data, with the goal of performing sophisticated tasks in an automated manner[58]. Within the Semantic Web, we could ask not only for the number of inhabitants of a country’s capital, but also we could ask for the nearest hospital to our current location that has facilities to take a given medical exam, or we could ask for relatives of political authorities who own companies that sell services to the government and the amount of invoices. In other words, if the vision of the Semantic Web were fully realized, we could ask complex queries and receive direct answers based on data automatically integrated from multiple sources on the Web.

2.3.1 RDF

The Resource Description Framework (RDF) is the framework used to represent information about *resources* – also called *entities* – in the Semantic Web. Resources can be anything, including documents, physical objects, abstract concepts, etc. RDF provides a way to publish machine-readable data, to link resources from different sources, to exchange data between applications, etc. [23].

2.3.2 RDF Data Model

The fundamental piece of information in RDF is called a *triple* (or *statement*). An RDF triple is a tuple of three elements: a subject, a predicate and an object and they are usually represented by the notation (s, p, o) . While we use the term *entity* (or resource) to talk about real-world objects, which usually appears in the *subject* or *object* position of an RDF triple, we use the term *property* to refer the elements appearing in the *predicate* position.

Specifically $(s, p, o) \in (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ where \mathbf{I} is the set of IRIs (Internationalized Resource Identifier)[28], \mathbf{B} is the set of blank nodes and \mathbf{L} is the set of literals. IRIs are used to identify resources, blank nodes are used to talk about a resource without naming it, and finally, literals are used to express datatyped values (e.g. dates, numbers, booleans) about resources. \mathbf{I} , \mathbf{B} and \mathbf{L} are disjoint sets known collectively as RDF terms [22, 23, 39].

RDF triples are usually represented by two nodes, the *subject* and the *object*, connected by a directed edge labeled with the *predicate*. They express that the *subject* is related by the *predicate* to the *object*. A set of RDF triples is called an *RDF graph*.

Example 2.8. *`http://www.wikidata.org/wiki/Q201674` is an IRI that names the resource (Q201674), `http://www.w3.org/2000/01/rdf-schema#label` is another IRI that names the property label, and “The Deer Hunter” is a literal. By ordering them in a triple we can say: (Q201674, label, The Deer Hunter) which means that the resource Q201674 has the label “The Deer Hunter”. Figure 2.4a represents how these three RDF terms together form an RDF triple.*

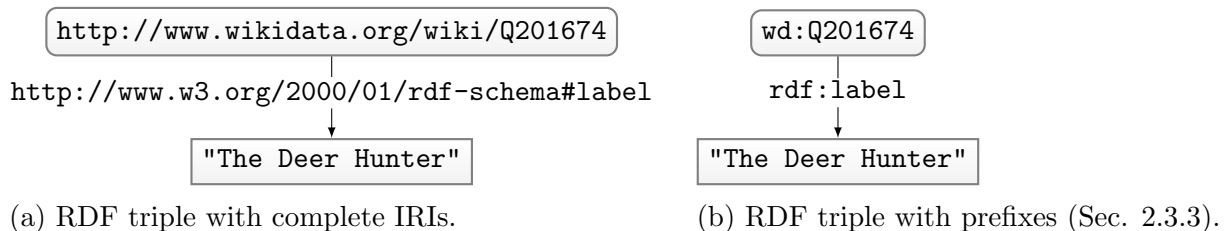


Figure 2.4: Two RDF triples making a statement about the movie “The Deer Hunter”. On the left, (a) the triple is represented with full IRIs. On the right, (b) the triple is represented using prefix shortcuts.

2.3.3 RDF Vocabularies

An RDF vocabulary is a collection of IRIs intended for use in RDF graphs [22]. They often begin with a common substring known as a namespace IRI. Some namespace IRIs are associated with a prefix. Table 2.1 shows some examples of RDF vocabularies. These prefixes are commonly used to abbreviate IRIs. For example, `http://www.wikidata.org/wiki/Q172241` is usually abbreviated as `wd:Q172241`. Note that *rdf* and *rdfs* (described later in Section 2.3.4) are standard RDF Vocabularies. Figure 2.4 represents twice an RDF triple using (a) complete IRIs and (b) RDF Vocabularies.

Table 2.1: Examples of Namespace prefix and IRIs. Based on [22]

Prefix	Namespace IRI	RDF Vocabulary
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	The RDF built-in vocabulary
rdfs	http://www.w3.org/2000/01/rdf-schema#	The RDF Schema vocabulary
xsd	http://www.w3.org/2001/XMLSchema#	The RDF-compatible XSD types
foaf	http://xmlns.com/foaf/0.1/	Friend of a Friend vocabulary
wd	http://www.wikidata.org/wiki/	Entities in Wikidata
owl	http://www.w3.org/2002/07/owl#	Web Ontology Language [10]

Example 2.9. *Let us codify the Example 2.7 in RDF. Instead of using full IRIs, we are going to use an example IRI as namespace: `http://www.ex.org`. Furthermore, we are going to use labels to name entities, and we are going to change the way of representing IRIs: instead of using the initials, we are going to use part of the name. Thus, we are going to write `ex:RDeNiro` instead of `http://www.ex.org/RDN`. Figure 2.5 shows the information from Example 2.7 encoded in RDF using Turtle syntax [11]. Figure 2.6 shows an actual graph representation from an extract of this RDF graph.*

```

ex:RDeNiro ex:name "Robert De Niro";
           ex:star ex:TaxiDriver, ex:DeerHunter, ex:RagingBull, ex:JackieBrown;
           ex:produce ex:Thunderheart, ex:IrishMan;
           ex:direct ex:BroxTale, ex:GoodShephard .
ex:CWalken ex:name "Christopher Walken";
           ex:star ex:DeerHunter .
ex:SLJackson ex:name "Samuel L. Jackson";
             ex:star ex:JackieBrown ;
             ex:produce ex:EveBayou .
ex:QTarantino ex:name "Quentin Tarantino";
              ex:direct ex:JackieBrown;
              ex:produce ex:DeathProof, ex:Hostel .
ex:SLelio ex:name "Sebastian Lelio" ;
          ex:direct ex:AFantasticWoman, ex:Gloria .
ex:PLarrain ex:name "Pablo Larrain" ;
            ex:produce ex:AFantasticWoman .
ex:DVega ex:name "Daniela Vega" ;
          ex:star ex:AFantasticWoman .
ex:JMoore ex:name "Julianne Moore" ;
          ex:star ex:Gloria ;
          ex:produce ex:MarieBruce .

```

Figure 2.5: An RDF graph. Part of the information from Example 2.7 encoded in RDF.

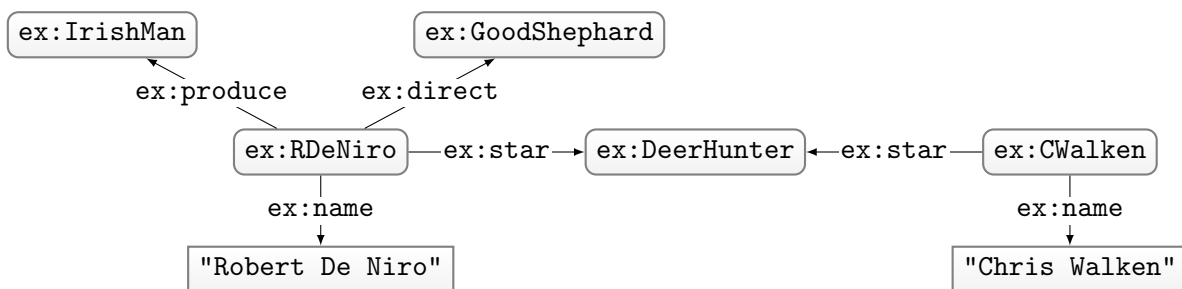


Figure 2.6: Graph representation of part of the RDF graph from Figure 2.5. We can see how `ex:RDeNiro` is indirectly related to `ex:CWalken` because they starred in the same movie, `ex:DeerHunter`.

2.3.4 RDF Schema

RDF Schema (RDFS) is a semantic extension of RDF. It provides mechanisms for describing groups of related resources and the relationships between these resources [15]. While RDF defines *rdf:type* and *rdf:Property* among others terms, RDFS defines *rdfs:Resource*, *rdfs:Class*, *rdfs:Literal*, *rdfs:subClassOf*, *rdfs:subPropertyOf*, *rdfs:domain*, *rdfs:range*, etc.

All things described by RDF are *rdfs:Resource*. A class is a set of resources that share some commonalities; an example might be *building*, or *person*, or *species*, etc. The class of resources that are classes is *rdfs:Class*. *rdf:Property* is the class of all properties, which are binary relations between a subject and an object. *rdfs:subClassOf* and *rdfs:subPropertyOf* are used to express hierarchies between classes and properties. Finally, *rdfs:domain* and *rdfs:range* are used to express restrictions about the classes of subjects and the objects related by a property.

Up to this point, we have reviewed the most basic topics of the Semantic Web. Despite their importance, we do not include further Semantic Web standards relating to RDF syntaxes or OWL (The Web Ontology Language designed to represent rich and complex knowledge about things, groups of things, and relations between things [10]) since they are not fundamental for the development of this work.

In the next section, we are going to cover *characteristic sets*, another cornerstone of our work. This technique generates a set of sets of properties from an RDF graph, where each set of properties is used to define a particular entity. We are going to use these sets to bridge from the world of Formal Concept Analysis to the Semantic Web, which will ultimately lead us to our proposed hierarchical schema for RDF.

2.4 Characteristic Sets

Cardinality estimation is the basis for query optimization. It allows database management systems to estimate how many results will be returned by subqueries of an input query, which is important to find an efficient execution order. In order to estimate the cardinality of a RDF graph, Neumann and Moerkotte [50] defined a new structure, called a *characteristic set*. Here we will use the same structure but with the goal of building a hierarchical schema in mind.

In order to define the characteristic sets, we will use a new operator: π_{\bullet} . Given an RDF graph G , for $\bullet \in \{s, p, o\}$, we denote by $\pi_{\bullet}(G)$ the projection of the set of terms appearing in a particular triple position in G ; e.g., $\pi_s(G) := \{s \mid \exists p, o : (s, p, o) \in G\}$. We also use this notation for more than one triple position, for example, $\pi_{s,p}(G) := \{(s, p) \mid \exists o : (s, p, o) \in G\}$.

Definition 2.13 (Characteristic Sets). *Let G be an RDF Graph and $(s, p, o) \in G$ be an RDF triple, i.e., $s \in \pi_s(G)$, $p \in \pi_p(G)$, and $o \in \pi_o(G)$. The characteristic set \mathcal{C} of an entity s is the set of its associated properties, i.e.: $\mathcal{C}(G, s) := \{p \in \pi_p(G) \mid \exists o \in \pi_o(G) : (s, p, o) \in G\}$, while the set of characteristic sets of an RDF graph is $\mathcal{C}(G) := \{\mathcal{C}(G, s) \mid s \in \pi_s(G)\}$ [50]. We further define $\mathcal{C}(G, s) := \emptyset, \forall s \notin \pi_s(G)$. We say that the size of a characteristic set is the number of properties that it contains, while its length is the number of subjects that share the same characteristic set.*

Remark 2.3. *Let $\llbracket G \rrbracket \subseteq 2^{\pi_s(G)} \times 2^{\pi_p(G)}$ denote a set such that:*

- $\bigcup_{(S, P) \in \llbracket G \rrbracket} S \times P = \pi_{s,p}(G)$
- $\forall (S, P) \in \llbracket G \rrbracket, S \neq \emptyset, \text{ and } P \neq \emptyset$
- $\forall (S, P), (S', P') \in \llbracket G \rrbracket, S \cap S' = \emptyset, \text{ and } P \neq P'$

In words, $\llbracket G \rrbracket$ is a set of pairs. We say that (S, P) is in $\llbracket G \rrbracket$ if and only if S is a set of entities such that its characteristic set is P .

Example 2.10. *Figure 2.7 represents $\llbracket G \rrbracket$, where G is the RDF graph of Figure 2.5*

<code>{ex:RDeNiro}</code>	<code>{ex:name, ex:star, ex:produce, ex:direct}</code>
<code>{ex:CWalken, ex:DVega}</code>	<code>{ex:name, ex:star}</code>
<code>{ex:SLJackson, ex:JMoore}</code>	<code>{ex:name, ex:star, ex:produce}</code>
<code>{ex:QTarantino}</code>	<code>{ex:name, ex:direct, ex:produce}</code>
<code>{ex:SLelio}</code>	<code>{ex:name, ex:direct}</code>

Figure 2.7: $\llbracket G \rrbracket$, where G is the RDF graph from Figure 2.5. It represent the set of entities (left column) that share a characteristic set (right column).

2.5 Wikidata

On the Web, a number of websites have become household names. Among the most prominent such websites, we can mention the top five according to *Alexa*¹: Google², a search engine[16]; Youtube³, a place to share videos; Facebook⁴, a social network[30]; Baidu⁵, a Chinese search engine; and Wikipedia⁶, a free online encyclopedia that anyone can edit[9].

Wikipedia allows users – called collaborators – to create, edit and publish documents. Once those documents are published, they can be read by anyone on the Web. Wikipedia was created in 2001. Since then, more than 47 million articles have been published in more than 290 languages⁷.

Being based on the Web (of Documents), Wikipedia was designed to be read by humans. Consequently, its content is barely machine-interpretable. Furthermore, it separates different languages, providing one Wikipedia for each dialect. Specifically, there is one Wikipedia in Spanish⁸, another in English⁹, another in German¹⁰, and so on.

To solve these and other problems, a *Semantic Wikipedia*, called *Wikidata*[47] was created. Wikidata can be seen as a Wikipedia for data that manages factual information in a multilingual way without losing the open editing and community control that are the distinguishing characteristics of Wikipedia. Today, Wikidata has more than 45 million pages, and it has been edited by more than 35 thousand editors¹¹.

Example 2.11. *Figure 2.8 show the webpage of Robert De Niro in Wikidata. We can see that Wikidata uses numeric identifiers (Q36949) to name entities. The statements can be read as “Q36949 instance of human”, “Q36949 sex or gender male”, and “Q36949 country of citizenship United States of America and Italy”. The picture does not show that not only “Robert De Niro”, “human”, “male”, “United States of America”, and “Italy”; but also “instance of”, “sex or gender”, and “country of citizenship” are actually entities in the same graph. Wikidata displays the label of the entity instead.*

¹<https://www.alexa.com/topsites>

²<https://www.google.com>

³<https://www.youtube.com>

⁴<https://www.facebook.com>

⁵<https://www.baidu.com>

⁶<https://www.wikipedia.org>

⁷https://meta.wikimedia.org/wiki/List_of_Wikipedias. Accessed on 28.01.2018

⁸<https://es.wikipedia.org>

⁹<https://en.wikipedia.org>

¹⁰<https://de.wikipedia.org>

¹¹<https://stats.wikimedia.org/v2/#/wikidata.org>. Accessed on 13.02.2018

Robert De Niro (Q36949)

American actor, director and producer

 edit

Robert, Jr. De Niro | Jr. Robert De Niro | Robert Anthony De Niro | De Niro

Statements






instance of	 human  edit ▶ 1 reference	+ add value
sex or gender	 male  edit ▶ 5 references	+ add value
country of citizenship	 United States of America  edit ▼ 0 references + add reference	
	 Italy  edit ▶ 1 reference	+ add value

Figure 2.8: Screenshot of Robert De Niro’s webpage in Wikidata.

Wikidata provides large and heterogeneous dumps of data on a weekly basis¹². While Wikidata makes different kind of data available and in several formats, for the experiments presented in this thesis, we are going to use the simplest one: *truthy* dumps in *N-Triples*[38] syntax.

Truthy statements are non-deprecated statements that don’t contain metadata such as qualifiers and references. Furthermore, if there is a preferred statement for a given property, only preferred statements for that property will be considered as *truthy*[3]. As an example, Wikidata stores the number of inhabitants in Chile in years 1960 (7,649,026), 1970 (9,578,923), 1980 (11,192,284), etc. Nevertheless, the unique *truthy* statement about the number of inhabitants in Chile is the most recent one with 17,619,708 people.

The Wikidata dumps are big enough to make it challenging to process them on a single commodity machine. For example, the largest dump we will work with in our evaluation contains 1.2 billion triples. Nevertheless, dumps of this size can be processed more quickly by several commodity machines in a distributed manner. In the next section, we are going to talk about *Hadoop*, a framework for distributed computing that we will use later to extract the characteristic sets from *truthy* dumps for various versions of Wikidata.

¹²https://www.wikidata.org/wiki/Wikidata:Database_download. Accessed on 13.02.2018

2.6 Apache Hadoop

Apache Hadoop is a framework for distributed processing of large data sets across clusters of commodity hardware in fault-tolerant manner. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage [1].

The most important parts of Apache Hadoop are Hadoop Distributed File System and Hadoop MapReduce.

2.6.1 Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is a distributed file system designed to be highly fault-tolerant and to run on commodity (low-cost) hardware [14]. HDFS assumes that hardware will fail. In order to avoid data loss, HDFS has automatic recovery processes.

A HDFS cluster consists of one or more NameNodes and potentially thousands of DataNodes working together in a master/slave architecture. The NameNodes act as master servers that manage the file system and regulate access to files by clients. On the other hand, DataNodes manage storage. When a file is added to HDFS, it is split into one or more blocks and these blocks are stored in a set of DataNodes.

To enable fault tolerance, these blocks may be replicated on a number of machines specified as a replication factor (e.g., 3). When a machine fails, the blocks that were stored on that machine are copied to another machine from a working replica to satisfy the replication factor. This process is fault tolerant so long as a working replica exists for each block (and there is sufficient space and machines to physically satisfy the replication factor).

2.6.2 Hadoop MapReduce

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data in parallel on large clusters of commodity hardware in a reliable, fault-tolerant manner [2].

MapReduce needs at least two tasks: a *map* task and a *reduce* task. The map task takes the input and produces an intermediary set of key-value pairs, which are automatically grouped by the framework according to their key, in a phase called *shuffle*. This phase produces for each unique key a list of values that share the same key. Afterwards, the reduce task takes, for each key, a stream of all its associated values, and applies some user-defined operation over that stream to produce an output [26].

Optionally, a user could define a combiner task, which runs between the map task and the reduce tasks. Its primary goal is to perform a reduce operation with data in the same node before sending data across the network.

Globally, MapReduce takes a set of key-value pairs as input and produces another set of key-value pairs as output. A complete iteration of MapReduce is known as a *MapReduce Job*. More complex jobs can be broken down into chains (or directed acyclical graphs) of MapReduce jobs, where the output of one or more jobs may serve as the input for one or more future jobs.

Hadoop MapReduce can be configured by several parameters. One of the most important parameters to improve performance of MapReduce is the amount of memory given to the *containers*. While Hadoop HDFS guarantee fault tolerance by splitting (and replicating) data in blocks among several machines, Hadoop MapReduce guarantees fault tolerance of computing by splitting (and tracking) processing into containers. There is a trade-off between the amount of memory for containers: the more memory a container has, the more data it can process, but the number of map tasks per machine will be reduced.

For the last section of this chapter introducing some necessary preliminaries, we are going change focus again. Once we analyze the RDF dumps provided by Wikidata with MapReduce, we are going to check if we can predict how the data evolve and as a baseline for that task, we are going to use a linear regression model.

2.7 Linear Regression Models

Regression analysis is a set of statistical processes for estimating the relationships among variables. In a typical scenario, we have an outcome measurement that we want to predict based on a set of features. We have a training set, where we observe the outcome (or *response*) and feature measurements (or *predictors*) for a set of objects. Using this data we build a prediction model, which will enable us to predict the outcome of new unseen objects[33].

Linear Regression Models are regression models that assume linear dependency between the response and its predictors. When there is just one predictor (Simple Linear Regression Model), we assume that the outcome y_i can be modeled as $\hat{y}_i = \hat{\beta}_0 + x_i\hat{\beta}_1$, where $\hat{\beta}_0$ is called *intercept* or *bias*, and $\hat{\beta}_1$ is called the coefficient. We use the notation $_i$ to name the objects in the training set, and $\hat{\cdot}$ to say that \cdot is an estimated value instead of a real value.

Given a training set, the error (or *residual*) is the difference between the real value (y_i) and the predicted value (\hat{y}_i). It is represented as $\hat{e}_i = y_i - \hat{y}_i$. In order to fit a linear model, we look for $\hat{\beta}_0$ and $\hat{\beta}_1$ such that minimize the sum of the squared errors (SSE). $SSE = \sum_i \hat{e}_i^2 = \sum_i (y_i - \hat{y}_i)^2$. This method is called *Least Squares*. After solving the optimization, the coefficients $\hat{\beta}_1$ and $\hat{\beta}_0$ are given by Equations 2.1, and 2.2 respectively.

$$\hat{\beta}_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} \quad (2.1)$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \frac{\sum (x_i)}{n} \quad (2.2)$$

When we fit a model, we use the least squares method over our training set. But there is no guarantee that the SSE observed in training will be the error when we predict over new objects. There are several indicators to evaluate how a regression model performs over new objects. Two of them are the Root Mean Square Error (RMSE) and the Mean Absolute Error (MAE). Both of them are given by Equations 2.3 and 2.4 respectively, where n is the number of observations in the training set.

$$RMSE := \sqrt{\frac{\sum_i (y_i - \hat{y}_i)^2}{n}} \quad (2.3)$$

$$MAE := \frac{\sum_i |y_i - \hat{y}_i|}{n} \quad (2.4)$$

If the prediction error for each object is constant, then $RMSE = MAE$. On the other hand, if the error for some objects is much larger than others, then $RMSE \gg MAE$. More specifically, RMSE is sensitive to the variance in the distribution of errors for different objects.

2.8 Summary

Now, we have covered different preliminaries that we will use during this work. We have reviewed (1) the mathematical foundations of our data-driven schema proposal, especially lattice theory, (2) the RDF data model, that allows the data to grow without any schema, (3) characteristic sets, our molecular structure to generate a data-driven schema, (4) Wikidata, which is a prominent RDF graph twinned with Wikipedia, (5) Apache Hadoop, a framework for distributed and scalable computation, and (6) the simple linear regression model that we are going to use to evaluate how our data-driven schema predicts the evolution of such RDF graph as Wikidata.

Chapter 3

Related Work

In this chapter, we cover four axes highly related to our work. First, we review some strategies to construct a formal concept lattice from a formal concept visiting different kind of algorithms: sequential, parallel and distributed. Second, we review how the formal concept analysis field has been applied in the Semantic Web community for several purposes, like extracting questions from RDF graphs, or formal concept discovery in Semantic Web data. Third, we review different approaches to extract disparate kinds of schema from an RDF graph; we start with approaches considering only relations between classes, then discussing strategies to generate a relational schema from RDF graphs, before finishing with a novel summarization framework based on node equivalence and quotient graphs. Finally, we review other approaches that looked to model dynamics on the Semantic Web.

3.1 Computing Formal Concept Lattices

After Wille published his Restructuring lattice theory in 1982 [62], several algorithms were proposed for generating a formal concept lattice of a given formal context; examples include: algorithms by Bordat [13], Ganter (or nextClosure) [35], Chein [20], Norris [51], Godin and Nourine [52]. An experimental comparison of these algorithms shows that Ganter has better performance for large and dense data. In their experiments, they generated random datasets with at most 1,000 different entities, which at the time was considered large.

Some years later, there was another resurgence of interest in algorithms for computing lattices, but using other paradigms. Fu and Nguifo [34] created a parallel version of the Ganter algorithm based on partitioning the search space for concepts. They show that its version can be 60 times faster than the sequential approach. They tested their implementation in a dataset of at most 50 attributes (or columns).

Krajca *et al.* [45] first developed a parallel recursive algorithm for formal concept analysis. One of the key features of their implementation is that the program simulated a sequential code until a point where the procedure forks into multiple recursive processes that compute disjoint formal concepts. For the evaluation, the biggest dataset they used contained 32,710

entities and 295 attributes. Afterward, Krajca *et al.* [46] transformed their previous implementation in order to make it distributed instead of just parallel. They used the MapReduce framework and evaluated the performance of the algorithms using the same dataset as before. While the performance increased, it is very dependent on the dataset structure, specifically on the density of the matrix (fraction of components distinct to zero).

Xu *et al.* [29] also developed distributed formal concept analysis algorithms, but based on a special branch of MapReduce, called *Twister*, which enhances the core of MapReduce by allowing iterations within MapReduce tasks. First, they modified the Ganter algorithm to make it distributed and then improved this new version by allowing iterations. The biggest dataset used in the evaluation process was composed of 103,950 entities and 133 attributes.

While the Krajca distributed algorithm works better with low-density matrices, the Xu algorithm works better with high-density matrices. Nevertheless, both of them evaluated their programs on relatively small-sized datasets (when compared with, e.g., Wikidata).

Regardless of the kind of implementation, the paradigm used, or the released time, the authors of these algorithms agree on one particular thing: computing a formal concept lattice is an important but computationally complex task. In the next section, we are going to review how these techniques have been applied for the Semantic Web.

3.2 Formal Concept Analysis on the Semantic Web

Formal Concept Analysis is a branch of applied mathematics that is used extensively in many research areas, including the Semantic Web.

Aquin and Motta [24] proposed a method and a tool to discover questions and answers from a given RDF graph. They built a hierarchy of formal concepts where the extent of the formal concepts was a set of instances and represented an answer, and the intent of the formal concepts was a set of properties and represented a query. Later, they used this hierarchy as a querying interface to propose questions letting the user specify the levels of granularity and specificity. While they tested their approach with four different datasets (geography, jobs, restaurants, drama) taken from a Ph.D. thesis [42], the datasets seem to have a tabular structure. On the other hand, the biggest graph (drama) contained 19,294 entities, i.e., it was not a big graph. Concordant with previously-discussed results, they had issues when the number of formal concepts was greater than a couple of thousand.

Other interesting work was developed by Formica [32] by mixing *rough set theory* with *fuzzy formal concept analysis* with the goal of performing Semantic Web search and discovery of information. One of the features of this proposal is that when a user requests data which is not modeled by any formal concept, the user can search and discover information interactively on the Web by moving to the sub or super formal concept in the lattice. However, a weakness of this work is the lack of an evaluation of their technique. They argue that due to the infeasibility of evaluating the match answers and the predefined queries in the formal concept lattice it is not possible to evaluate the whole methodology.

Kirchberger *et al.* [44] also applied formal concept analysis in Semantic Web data. They show a performance analysis of different algorithms to compute formal concepts. In their experiments, they considered nine RDF graphs encompassing from 316 to 31,936 entities, and from 119 to 20,707 properties. They show that for those RDF graphs – which are small compared to Wikidata – it is feasible to compute the formal concepts. Afterward, they perform a performance analysis varying the number of entities from 10,000 to 750,000 while maintaining the number of properties equal to 1,552. They report time-out and out-of-memory errors when they use bigger datasets.

Alam *et al.* [7] also use a concept lattice, but to organize information and automatically detect missing information. Their method consists in building a concept lattice of RDF datasets through the use of pattern structures (an extension of formal concept analysis able to process complex data descriptions) and then use this lattice to discover implication rules (association rules whose confidence is 100%). Within this new construction (lattice of formal concepts associated with implication rules), they were able to discover missing information and tackle the data completeness problem. Although they tested their approach in DBpedia, they didn't use the full dataset. They selected specific subsets (cars, video games, smartphones, countries) and executed their technique over them instead.

To the best of our knowledge, we have shown that all authors have had difficulties extracting formal concepts and building lattices with them. These kind of issues include both: general algorithms and specific evaluations in the Semantic Web. We have also shown that the common way to avoid this problem has been to select smaller datasets. On the other hand, our strategy is to not materialize the entire formal concept structure, but rather construct an intermediate structure based on characteristic sets; we will discuss our proposal further in Chapter 4. Beforehand, in the following section, we discuss other proposals for computing data-driven schemas from RDF graphs.

3.3 Data-Driven RDF schema

Regardless of the motive, several works have been conducted in order to compute a schema from an RDF graph.

Kinsella *et al.* [43] published one of the first works related to data-driven RDF schemas. They show the relation between classes of entities to help creators of ontologies [10] to reuse vocabulary and to monitor usage of classes and properties. They published a tool that was able to show links between instances of classes among several RDF datasets.

Dau and Sertkaya [25] applied formal concept analysis to cluster sets of objects along attributes and created a hierarchy of those clusters. They visualized the cluster hierarchy with Hasse diagrams to provide a new way to explore data.

In order to help users to formulate complex SPARQL¹[56] queries, Campinas *et al.* [17] generated a summary of an RDF dataset. They also argued that one of the principal reasons

¹SPARQL is the query language for RDF.

for the slow adoption of SPARQL is that the data is very diverse and does not follow any predefined structure or vocabulary. In their method, they defined a *data graph summary* as a “node collection”, which is a cluster of nodes considering two characteristics: class-based and attribute-based.

Abedjan and Naumann[5] applied *association rules*[33] for showing dependencies of schema elements among entities, schema discovery, ontology re-engineering, among other things. They grouped over a subject, predicate or object (according to the position of the RDF term in an RDF triple), one of the two remaining RDF terms, and executed association rules. When grouping predicates over subjects they found rules (among sets of predicates) that were used to discover schemas. Nevertheless, they did not propose a full schema for an RDF graph. They used DBpedia [8] as a data source.

A year after their previous paper, Abedjan and Naumann *et al.* [4] extended their work by developing a profiler for RDF data, called *ProLOD++*². With this tool, they were able to generate meta-data automatically for RDF datasets. Specifically, they identified unique combinations of predicates that identified entities and dependencies among subjects. They also suggested ontologies and detected mismatches of data and ontologies.

Pham *et al.* [54] derived a relational schema from RDF datasets in five steps: (1) Compute the characteristic sets of the RDF graph. (2) Label the characteristic sets using semantic information like ontologies. (3) Merge similar characteristic sets to make the schema more compact. (4) Filter out characteristic sets with low frequency. And finally, (5) filter out instances to have a cleaner schema (literal type homogeneity, multi-valued attributed, etc). They used this technique over DBpedia[8] and WebDataCommons[48] datasets and they found that 90% of RDF triples follow a tabular structure. Additionally, they discuss how the notion of “schema” differs from relational databases community (schema first) to graph databases community (schema last).

Čebirić *et al.* [18], discussed the characteristic that a graph summary should have. They focused on partially explicit and partially implicit RDF graphs. In order to test the characteristics, they computed a summary of an RDF graph and showed that their summary was helpful to code SPARQL queries.

Pham *et al.* [53] used the relational schema found in previous work[54] to improve efficiency of RDF systems. They targeted three problems: (1) the high execution cost of self-joins, (2) the low quality of SPARQL query optimization, and (3) the lack of options for optimizing RDF systems. Using a column-based database, they stored up to 95% of the data in the relational schema. The remaining 5% was stored in a table with three columns: predicate, subject, object. They were able to show a 3–10× overall speed improvement.

In her Ph.D. thesis, Čebirić *et al.* [19], defined an RDF-specific summarization framework (based on previous work[18]) based on *node equivalence* and *quotient graphs*[55]. She shows that, under certain conditions, an RDF graph can be efficiently summarized without materializing its implicit triples.

While all the previous work had proposed several ways to generate a data-driven schema

²<https://hpi.de/naumann/projects/data-profiling-and-analytics/prolod.html>

for RDF graphs, they do it statically without considering the inherently dynamic nature of the Semantic Web. In the next section, we are going to review some efforts to register and measure the changes in RDF graphs.

3.4 Modelling dynamics on the Semantic Web

Over the past few years, several authors have made efforts to record and measure the changes in RDF graphs over time.

Umbrich *et al.* [61] investigated how RDF datasets changed, and propose some measures to assess the changes. They argue that these kind of measurements are useful for tasks like (a) web crawling and caching, (b) distributed query optimization, (c) maintaining link integrity, (d) servicing of continuous queries, and (e) replication and synchronization. They not only defined dynamic metrics applicable in the Semantic Web, but also compared them with the metrics used in the Web of documents. Furthermore, they present a range of statistics on their findings. Nevertheless, due to its importance, they encourage further work on the area.

Three years after, Käfer *et al.* [41] created the Dynamic Linked Data Observatory project: a long-term experiment to monitor the documents in the Semantic Web on a weekly basis. Their methodology consisted of monitoring and retrieving 86,696 RDF documents and crawling its two-hop neighborhood. They reported several statistics on a document level (availability, death rate, change ratio), and on an RDF level (triples, terms, predicates, links) for 29 weeks. Among the possible uses of this kind of studies, they argue that measuring dynamics could be used to improve synchronization, caching, maintenance, versioning, even architectures.

From another point of view, focusing on the efficiency of systems in general (networks and computers), both Fernández *et al.* [31] and Divino *et al.* [27] investigated and surveyed several techniques to crawl RDF datasets. While the motivation of Fernández was to archive dynamics of the Semantic Web, the motivation of Divino was to maintain local repositories up to date.

3.5 Summary

Up to this point we have reviewed: (1) algorithms to extract formal concepts and build lattices, (2) how these techniques have been used in the Semantic Web, (3) other schema proposals for RDF, and (4) some efforts to measure the dynamics of RDF datasets. In the next chapter, we are going to define our schema proposal for large and diverse RDF graphs, inspired by formal concept analysis with the goal of measure and predict high-level changes in RDF graphs.

Chapter 4

Proposed Graph Schema

What is a *schema*? In its Greek definition, schema (schemas or schemata in their plural form) means *shape* or *plan*. It is *something* helpful to understand and give an order to a set of different objects. While this word can be used in several areas with different meanings, in computer science and especially in databases, a database schema is the logical data structure used to structure a given data set which is usually much smaller than the dataset [6].

On the other hand, lattices (Definition 2.9), and partial order sets in general (Definition 2.6) are algebraic structures that implicitly assert order among their elements through the partial order relation. In other words, they structure their elements. Subsequently, if we can define a partial ordered set from an RDF graph, such that (1) it is smaller than the data contained in the RDF graph, and (2) it represents and summarizes the data, then we could build a lattice and use this lattice as a hierarchical data-driven schema.

In this chapter, we describe three related schema proposals for RDF graphs: *RDF Concept Lattice*, *Characteristic Set Lattice*, and *Characteristic Set #-Lattice*. As their names say, all of them are lattices, but they differ in their elements, and consequently, in how they are ordered. An RDF Concept Lattice is composed of formal concepts (Definition 2.10) where the extents are sets of entities and the intents are sets of properties, a Characteristic Set Lattice is composed of characteristic sets (Definition 5.1) annotated with the set of entities having that characteristic set, and a Characteristic Set #-Lattice is composed of characteristic set annotated with the number of entities having that characteristic set.

4.1 RDF Concept Lattice

If we can define a formal context from an RDF graph, we could extract its formal concepts and generate a lattice with them (See Remark 2.2) which can be seen as a data-driven schema. To do so, we use the operator defined in Section 2.4: π_{\bullet} to select a projection of $\bullet \in \{S, P, O\}$ from an RDF graph G .

Let G be an RDF graph. A formal context X from G is a triple $X = (E, A, I)$, where $E := \pi_s(G)$, $A := \pi_p(G)$, and $I := \{(s, p) \mid \exists o : (s, p, o) \in G\}$. By applying Remark 2.2, we say that $\mathcal{L} := (\mathbb{C}, \leq)$ is a formal concept lattice from RDF graph G where \mathbb{C} is the set of all formal concepts of X and \leq the subset-superset relation given in Definition 2.12. Our strategy to compute a data-driven schema from an RDF graph is then:

1. Compute the Formal Concepts of G .
2. Generate the Concept Lattice.
3. Use the Concept Lattice as a schema for G .

Example 4.1. In this example we will construct an RDF Concept Lattice. Let us consider the RDF graph G from Figure 4.1. Figure 4.2 shows both, (1) the set of entities E , and (2) the set of attributes A in G . Figure 4.3 shows I from G , i.e., the binary relation between E and A , i.e., each tick means that an entity has an attribute. Figure 4.4 shows the formal concepts extracted from the previous cross table. And Figure 4.5 shows the Hasse diagram built with those formal concepts and the sub-, super-concept relation. Note that in the Hasse diagram we use only the first letter of the entities and properties.

```
ex:AK ex:director ex:Ikiru .
ex:AK ex:director ex:Ran .
ex:AK ex:name "A Kurosawa" .
ex:CE ex:director ex:Sully .
ex:CE ex:name "C Eastwood" .
ex:CE ex:star ex:Unforgiven .
ex:CE ex:star ex:Tightrope .
ex:GO ex:name "G Orwell" .
ex:GO ex:writer ex:1984 .
ex:PD ex:name "PK Dick"
ex:PD ex:writer ex:Ubik .
ex:PD ex:writer ex:Valis .
ex:UT ex:name "U Thurman" .
ex:UT ex:star ex:Gattaca .
```

Figure 4.1: An RDF example in Turtle Syntax.

- (1) $\{\text{ex:AK}, \text{ex:CE}, \text{ex:GO}, \text{ex:PD}, \text{ex:UT}\}$
- (2) $\{\text{ex:director}, \text{ex:name}, \text{ex:star}, \text{ex:writer}\}$

Figure 4.2: Set of (1) entities, and (2) properties from Figure 4.1.

	ex:AK	ex:CE	ex:GO	ex:PD	ex:UT
ex:director	✓	✓			
ex:name	✓	✓	✓	✓	✓
ex:star		✓			✓
ex:writer			✓	✓	

Figure 4.3: Cross table representing the formal context of Figure 4.1.

```

({}, {ex:director, ex:name, ex:star, ex:writer})
({ex:CE}, {ex:director, ex:name, ex:star})
({ex:AK, ex:CE}, {ex:director, ex:name})
({ex:CE, ex:UT}, {ex:name, ex:star})
({ex:GO, ex:PD}, {ex:name, ex:writer})
({ex:AK, ex:CE, ex:GO, ex:PD, ex:UT}, {ex:name})

```

Figure 4.4: Formal concepts from Figure 4.3.

Henceforth, we are going to use d , n , s , and w to denote `ex:director`, `ex:name`, `ex:star`, and `ex:writer` respectively. Analogously, we are going to use A , C , G , P and U to denote `ex:AK`, `ex:CE`, `ex:GO`, `ex:PD`, and `ex:UT`.

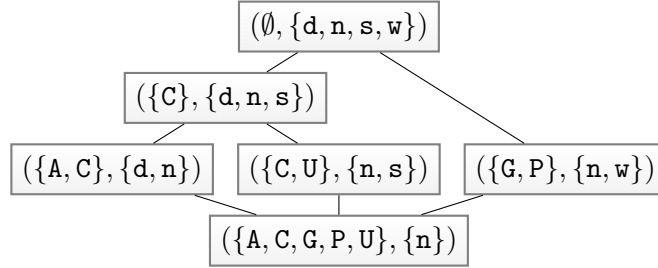


Figure 4.5: Hasse diagram of formal context derived from Figure 4.3. Nodes represent formal concepts and edges the direct sub-, super-concept relation (see Definition 2.12).

Intuitively, the lattice from Figure 4.5 represents a concept hierarchy distinguishing sets of entities based on the properties by which they are defined; for example, we can see concepts in the lattice relating to directors, actors, writers, and director–actors. Note that we maintain attribute subsets with the same cardinality on the same “level”. This is only a representation because neither lattices nor Hasse diagrams include the notion of a “level”. Note also, we only show the direct subset inclusion, as typical for Hasse diagram.

Mainly because formal concept analysis considers relations between only two sets: entities and properties, we did not consider the objects ($\pi_o(G)$) to build the lattice. We could annotate the properties with the objects, but then the resulting schema could be as big as the whole dataset, which is not a desirable situation.

This approach has some potential practical problems. One of those is that the number of formal concepts can be exponential ($\min(2^{|E|}, 2^{|A|})$) in the worst case. Several authors have

reported difficulties when computing formal concepts from different data sources, or simply they have used smaller datasets [21, 24, 63]. Another problem is that formal concepts include the same entities several times, which could make our schema even bigger than the original dataset. If we consider a big and heterogeneous RDF graph such as Wikidata, where the largest (most recent) dump we use in our experiments contains 57,197,406 entities and 3,492 attributes, it may not be practical to compute the RDF concept lattice.

For those reasons, we not only present a data-driven schema proposal based on formal concepts, but also two others based on the notion of characteristic sets.

4.2 Characteristic Set Lattice

Let G be an RDF graph, $\llbracket G \rrbracket$ be the set defined in Remark 2.3, i.e., $(S, P) \in \llbracket G \rrbracket$ if and only if S is a set of entities having the characteristic set P . Here we further define $\llbracket P \rrbracket_G = S$ such that $(S, P) \in \llbracket G \rrbracket$ (or $\llbracket P \rrbracket_G = \emptyset$ if no such S exists), and $\llbracket S \rrbracket_G = P$ such that $(S, P) \in \llbracket G \rrbracket$ (or $\llbracket S \rrbracket_G = \emptyset$ if no such P exists). We now define $\llbracket G \rrbracket^*$ (an extension of $\llbracket G \rrbracket$) as $\llbracket G \rrbracket^* := \llbracket G \rrbracket \cup \{(\emptyset, \emptyset), (\llbracket \pi_P(G) \rrbracket_G, \pi_P(G))\}$. Both (\emptyset, \emptyset) , and $(\llbracket \pi_P(G) \rrbracket_G, \pi_P(G))$ are going to be the bottom and the top of our schema. If we abuse the notation and define $(S, P) \subseteq (S', P')$ iff $P \subseteq P'$, then $\mathcal{L} := (\llbracket G \rrbracket^*, \subseteq)$ is called the Characteristic Set Lattice. In words, \mathcal{L} is a lattice of characteristic sets annotated with their subjects. Note that we are not considering the relation among the sets of entities anymore.

Example 4.2. Consider the RDF graph G from Figure 4.1. Figure 4.6 shows $\llbracket G \rrbracket^*$, while Figure 4.7 shows the Hasse diagram of $\mathcal{L} := (\llbracket G \rrbracket^*, \subseteq)$.

```
({}, {ex:director, ex:name, ex:star, ex:writer})
({ex:CE}, {ex:director, ex:name, ex:star})
({ex:AK}, {ex:director, ex:name})
({ex:UT}, {ex:name, ex:star})
({ex:GO, ex:PD}, {ex:name, ex:writer})
({}, {})
```

Figure 4.6: $\llbracket G \rrbracket^*$ derived from RDF graph in Figure 4.1.

Comparing the formal concepts presented in Figure 4.4 with the elements in $\llbracket G \rrbracket^*$ presented in Figure 4.6, we can notice that the entities appear only in one element from $\llbracket G \rrbracket^*$. This is a great advantage because it reduces the size in memory of the schema. But, we needed to re-define the partial order relation in order to only consider the set of properties. The fact that both of them have the same cardinality is just a coincidence.

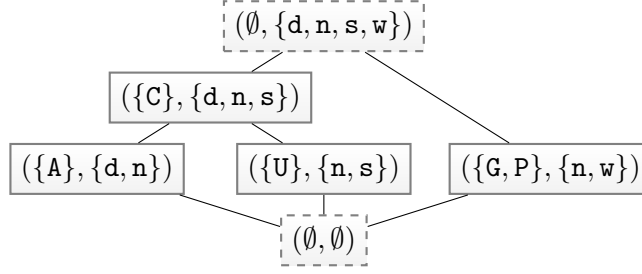


Figure 4.7: Hasse diagram of the Characteristic Set Lattice. The nodes represent characteristic sets annotated with subjects, while the edges represent the subset (\subseteq) relation between characteristic sets regardless the subjects. The bottom and top nodes are added to ensure a lattice with an infimum and supremum.

Note that if we would not have extended $\llbracket G \rrbracket^$ to include both the top and the bottom of \mathcal{L} , then we would not have a proper lattice.*

4.3 Characteristic Set #-Lattice

In order to reduce even more the size of our data-driven schema proposal, we avoid annotating the characteristic sets (P) with the set of entities having that characteristic set (S) by annotating them with only its cardinality ($|S|$). To do so, we need more definitions.

Let G be an RDF graph, $\llbracket G \rrbracket$ as defined in Remark 2.3, and $\llbracket G \rrbracket^*$ and \subseteq as defined in last section. We now define $\llbracket G \rrbracket^\# := \{(n, P) \mid \exists S : (S, P) \in \llbracket G \rrbracket^*, n = |S|\}$. In words, $(n, P) \in \llbracket G \rrbracket^\#$ if and only if n is the cardinality of the set of entities having the characteristic set P . We say then that $\mathcal{L}^\# := (\llbracket G \rrbracket^\#, \subseteq)$ is the Characteristic Set #-Lattice of the RDF graph G . We also denote by $\llbracket P \rrbracket_G^\# = |\llbracket P \rrbracket_G|$ the number of subjects that P has.

Example 4.3. Consider the RDF graph G from Figure 4.1. Figure 4.8 shows $\llbracket G \rrbracket^\#$, while Figure 4.9 shows the Hasse diagram of $\mathcal{L}^\# := (\llbracket G \rrbracket^\#, \subseteq)$.

```
(0, {ex:director, ex:name, ex:star, ex:writer})
(1, {ex:director, ex:name, ex:star})
(1, {ex:director, ex:name})
(1, {ex:name, ex:star})
(2, {ex:name, ex:writer})
(0, {})
```

Figure 4.8: $\llbracket G \rrbracket^\#$ derived from RDG graph in Figure 4.1.

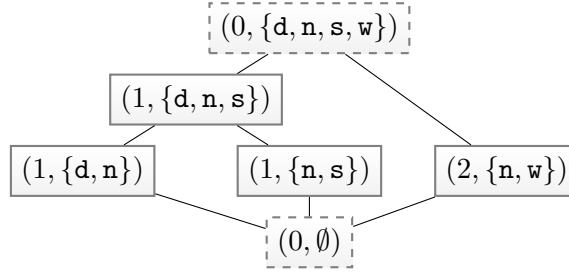


Figure 4.9: Hasse diagram of the Characteristic Set $\#$ -Lattice. The nodes represent characteristic sets annotated with the number of subjects having that characteristic set, while the edges represent the subset (\subseteq) relation between characteristic sets regardless of the number of subjects. As before, the bottom and top nodes are added to ensure a lattice with an infimum and supremum.

In the next chapter, we are going to use this lattice to define an algebra of lattices. We will define a *diff* operator between lattices, and an *add* operator between a lattice and a *diff*. Using these two operators, we will focus on the use case of predicting how the schema of an RDF graph evolves in time.

Chapter 5

Modelling Dynamics in RDF Graphs

We could intuitively consider the RDF Concept Lattice (or Characteristic Set Lattice) as encoding the possible paths of evolution of entities in an RDF graph: we could consider new entities as beginning at the bottom of the lattice (when they don't have properties) and evolving towards the top of the lattice as new properties are added to them, going from an incomplete data state towards the entity's full specification.

Referring back to Figure 4.5, for instance, we could consider new entities as first having *ex:name* defined, where they can then take a path towards being a director, an actor, or a writer; if already an actor or director, they may become an actor-director, and so forth.

Taking this one step further, if we have the lattices for two different points in time (dates, versions, etc.) of an RDF graph, we can apply a diff to see high-level changes between both versions of the data. Furthermore, given such a diff between two versions, we could further consider adding that diff to the most recent version to try predict future changes. We now capture precisely these intuitions with an algebra for computing diffs between lattices and adding diffs to lattices.

5.1 Characteristic Set Lattice Diff

Let G be an RDF graph such that we have access to different versions of it in a equally time distributed manner, for example, on a weekly or monthly basis. Let G_i and G_j be two consecutive versions of G such that $i < j$. Let $\mathcal{L}_i = (\llbracket G_i \rrbracket^*, \subseteq)$ and $\mathcal{L}_j = (\llbracket G_j \rrbracket^*, \subseteq)$ be the Characteristic Set Lattices for G_i and G_j .

We define the diff between \mathcal{L}_j and \mathcal{L}_i as $\Delta_{j,i} := \{(\mathcal{C}(G_j, s), s, \mathcal{C}(G_i, s)) \mid s \in \pi_s(G_i) \cup \pi_s(G_j)\}$. In simple words, $\Delta_{j,i}$ is a set of triples, where the second element is an entity s , while the first and the third are the characteristic sets that s has in versions j and i respectively. When $\mathcal{C}(G_j, s) \neq \mathcal{C}(G_i, s)$ we say that s *moves* from $\mathcal{C}(G_i, s)$ to $\mathcal{C}(G_j, s)$. In contrast, if $\mathcal{C}(G_j, s) = \mathcal{C}(G_i, s)$, then we say that s *stays* or *remains* in the same characteristic set.

Note that $\Delta_{j,i}$ also represent a graph. In this case, the nodes are the characteristic sets and the edges are the moving entities. Note also that when an entity is created, i.e. $s \in \pi_s(G_j)/\pi_s(G_i)$, by definition $\mathcal{C}(G_i, s) = \emptyset$. In the same way, when an entity is deleted, i.e., $s \in \pi_s G_i/\pi_s G_j$, then $\mathcal{C}(G_j, s) = \emptyset$. In both cases, we are considering the bottom of the lattice – (\emptyset, \emptyset) – as an infinite source of new and deleted entities. Example 5.1 shows two Characteristic Set Lattices, corresponding to two different versions and the diff computed from them.

Example 5.1. Figure 5.1 shows an example of the diff computed from two Characteristic Set Lattice, \mathcal{L}_2 and \mathcal{L}_1 , where \mathcal{L}_1 is the Characteristic Set Lattice previously introduced in Figure 4.7 and \mathcal{L}_2 is one possible way of how the lattice evolves in the next versions of the dataset. The diff is then a directed edge-labeled graph where the nodes are the sets of characteristic sets and the edges are labeled according to the entities that move between the nodes from version 1 to version 2.

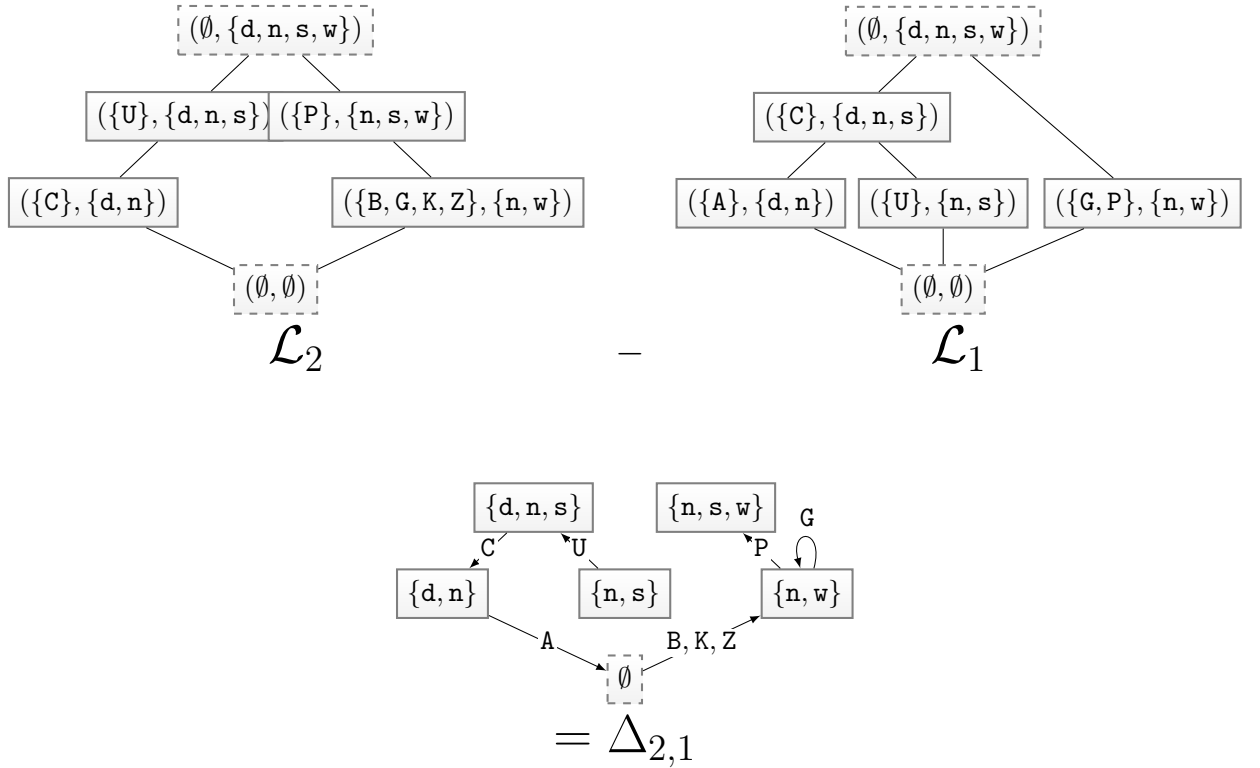


Figure 5.1: Diff $\Delta_{2,1}$ of two Characteristic Set Lattices \mathcal{L}_2 and \mathcal{L}_1 .

While this diff is useful to check how entities move between characteristic sets in different versions of a dataset, it is not very useful to model high-level changes. To avoid this issue, and with the goal of summarizing high-level changes, we also define a cardinality version of the diff: $\Delta_{j,i}^\# := \{(P', n, P) : n = |\{s : (P', s, P) \in \Delta_{j,i}\}|\}$. In words, n is the number of entities moving from characteristic set P in version i of the graph, to characteristic set P' in version j . In addition, we say that $\Delta_{j,i}^\#(P', P) = n$ as a way to get the number of entities moving directly. If no entity moves from P to P' , then $\Delta_{j,i}^\#(P', P) = 0$.

Example 5.2. Figure 5.2 shows the cardinality $\Delta_{2,1}^\#$ from $\Delta_{2,1}$ in Figure 5.1, i.e., It shows the total amount of moving entities.

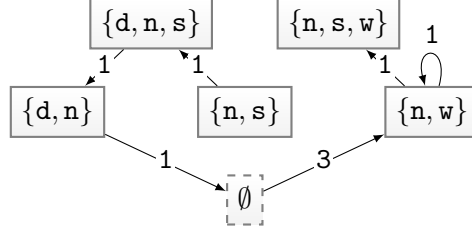


Figure 5.2: Cardinality diff from Figure 5.1. Here the list of entities moving was replaced with the total amount of entities.

5.2 Adding a Diff to a Lattice

Given three Characteristic Set Lattices (see Section 4.2) \mathcal{L}_i , \mathcal{L}_j , and \mathcal{L}_k , referring to three versions of an RDF graph (we assume that $i < j \leq k$) we could consider adding the diff $\Delta_{j,i} = \mathcal{L}_j - \mathcal{L}_i$ to the lattice \mathcal{L}_k to predict a future version of the dataset through an operation such as $\mathcal{L}_{[k,j,i]} = \mathcal{L}_k + \Delta_{j,i}$. However, this operation does not make much sense because it details the movement of specific entities from one characteristic set to another, i.e., it will predict that an entity e will move again between the same characteristic sets.

To avoid this issue, we can consider predicting the Characteristic Set $\#$ -Lattice (see Section 4.3) in a operation such as $\mathcal{L}_{[k,j,i]}^\# = \mathcal{L}_k^\# + \Delta_{j,i}^\#$, i.e., to sum the incoming entities and subtract the outgoing entities for each characteristic set between versions i and j and add that total to version k ; for example, let us say that n entities move from some characteristic set $\{p, q\}$ in version i to $\{p, q, r\}$ in version j ; then starting with $\mathcal{L}_k^\#$, we could add n to the number of entities for $\{p, q, r\}$ and remove n from $\{p, q\}$ when computing $\mathcal{L}_{[k,j,i]}^\#$. But what if $\mathcal{L}_k^\#$ does not have n entities in the source characteristic set $\{p, q\}$ to move to $\{p, q, r\}$? Furthermore, what if more entities should move from $\{p, q\}$ to another set $\{p, q, s\}$?

To resolve such issues, rather than adding and subtracting the absolute numbers of entities, we first compute the ratio of entities that move from the source characteristic set, to then sum up all incoming ratios weighted with the number of entities in the source characteristic set. Formally, given a diff $\Delta_{j,i}^\#$ we define the ratio of subjects moving from P to P' by Equation 5.1.

$$\rho_{j,i}(P', P) := \begin{cases} \frac{\Delta_{j,i}^\#(P', P)}{\llbracket P \rrbracket_{G_i}^\#}, & \text{if } \llbracket P \rrbracket_{G_i} \neq \emptyset \\ 1, & \text{if } \llbracket P \rrbracket_{G_i} = \emptyset \text{ and } P = P' \\ 0, & \text{if } \llbracket P \rrbracket_{G_i} = \emptyset \text{ and } P \neq P' \end{cases} \quad (5.1)$$

In the first case, we divide the number of entities moving from P (in version i) to P' (in version j) by the total number of entities in P . Since the diff between versions j and i can be applied to another version k , it is possible that there are entities in a characteristic set of version k for which there are no entities in version i (formally, $[[P]]_{G_i} = \emptyset$, which would lead to a divide-by-zero in the first case of Equation 5.1). When this happens, in the absence of further information, we assume that the source entities in the corresponding characteristic set of k will stay where they are; this is defined by the latter two cases in Equation 5.1. Finally, we then define $\mathcal{L}_{[k,j,i]}^\# = \mathcal{L}_k^\# + \Delta_{j,i}^\# := (\{(\sigma(P), P) \mid P \neq \emptyset \text{ and } \sigma(P) \neq 0\}, \subseteq)$ where $\sigma(P)$ is defined by Equation 5.2.

$$\sigma(P) := \text{round} \left(\sum_{P_k \in \{P \mid \exists S: (S, P) \in [[G_k]]\}} \rho_{j,i}(P, P_k) \times [[P_k]]_{G_k}^\# \right) + \Delta_{j,i}^\#(P, \emptyset). \quad (5.2)$$

The summand $\Delta_{j,i}^\#(P, \emptyset)$ adds the number of fresh subjects (not appearing in version i) added to P in version j . Finally, we add top and bottom concepts (as before) to $\mathcal{L}_{k,j,i}^-$ to generate the predicted #-lattice $\mathcal{L}_{k,j,i}^\#$.

Example 5.3. Figure 5.3 shows an example of adding a #-diff to a #-lattice to predict the next #-lattice (with $\mathcal{L}_2^\#$ and $\Delta_{2,1}^\#$ based on Figure 5.2). Take the case of $\{\mathbf{n}, w\}$: in $\mathcal{L}_2^\#$ this characteristic set has 4 entities, of which, $\Delta_{2,1}^\#$ states that half (2) should stay in $\{\mathbf{n}, w\}$ while half (2) should go to $\{\mathbf{n}, s, w\}$; furthermore, 3 fresh entities are defined for $\{\mathbf{n}, w\}$; hence the predicted value for $\{\mathbf{n}, w\}$ is 5. Consider on the other hand $\{\mathbf{n}, s, w\}$: in $\Delta_{2,1}^\#$ it has no outgoing edges since it was not present in \mathcal{L}_1 , hence the one entity in $\mathcal{L}_2^\#$ remains and 2 are added from $\{\mathbf{n}, w\}$ as aforementioned; thus the predicted value is 3. Finally, we highlight that $\{\mathbf{d}, \mathbf{n}, s\}$ is predicted empty: though $\Delta_{2,1}^\#$ suggests that entities should be added from $\{\mathbf{n}, s\}$, no such entities are available in $\mathcal{L}_2^\#$, and the entity previously in $\{\mathbf{d}, \mathbf{n}, s\}$ moves to $\{\mathbf{d}, \mathbf{n}\}$ (while the previous entity in $\{\mathbf{d}, \mathbf{n}\}$ is deleted, leaving one entity in $\{\mathbf{d}, \mathbf{n}\}$).

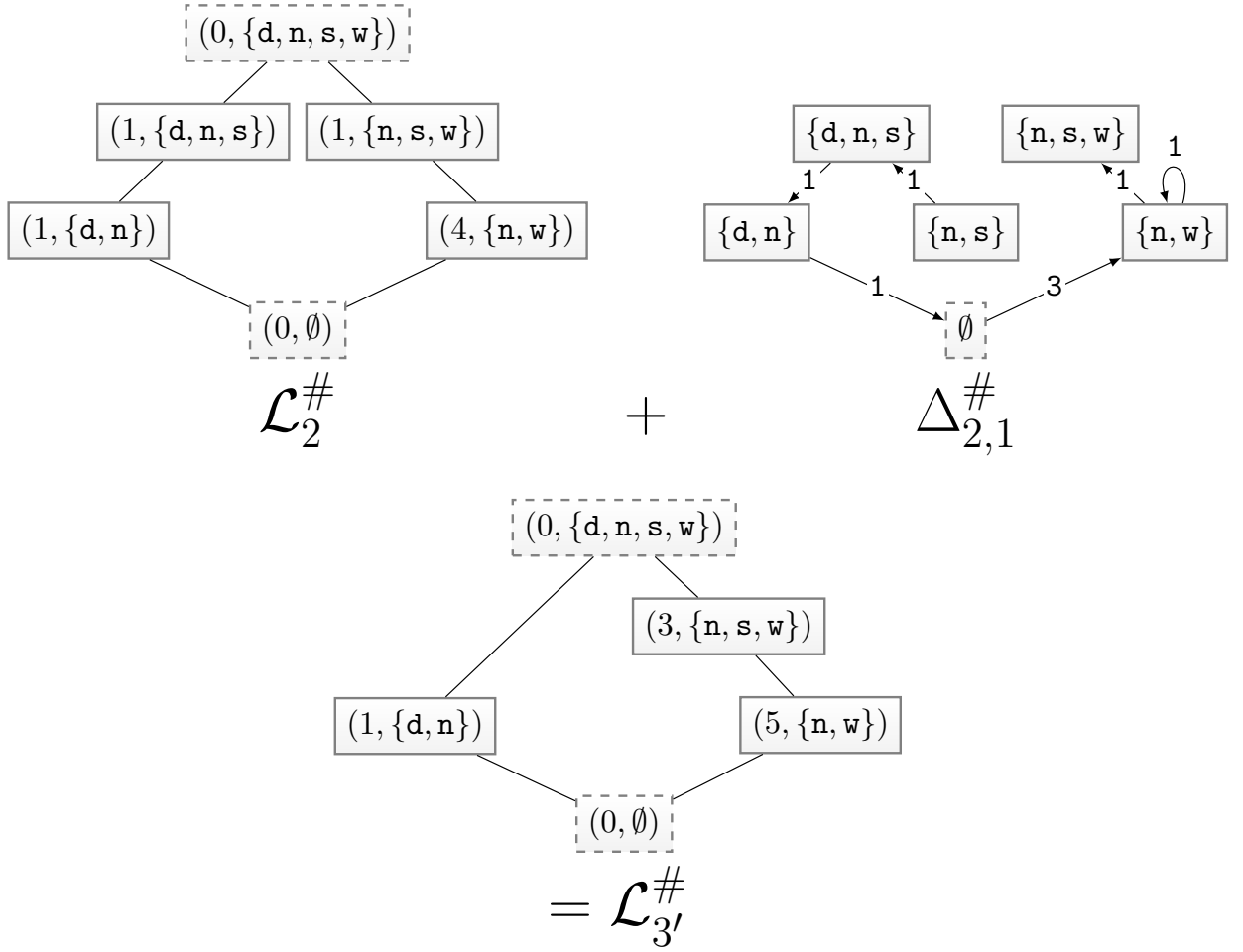


Figure 5.3: Adding a cardinality diff to the most recent Characteristic Set $\#$ -Lattice to predict the next Characteristic Set $\#$ -Lattice.

With these algebraic operations, we are able to predict future high-level changes in RDF graphs. Note that if we have access to more than two versions of an RDF graph, we could compute the diffs between consecutive pairs of versions, and average them in order to generate one single diff to predict a new version. This might, for example, smooth the effect of bulk edits performed by bots, where large numbers of entities transition between characteristic sets as a once-off event.

In the next chapter, we will begin to describe our implementation of the proposed methods. First, we will show a practical way to extract the characteristic sets of an RDF graph that scales. Later we will propose algorithms to compute a lattice from the characteristic sets extracted in the previous step.

Chapter 6

Extraction of Characteristic Sets

In this chapter, we are going to describe an empirical evaluation of how we extracted the characteristic sets from the Wikidata RDF graph. Given the scale of the graph, we explore extraction methods distributed over multiple machines based on the MapReduce framework. First, we are going to describe the base of MapReduce algorithms used to compute the characteristics sets at scale. After that, in order to improve efficiency, we propose various optimizations on top of those base algorithms and present performance experiments.

6.1 Architecture

We used two different environments to run our experiments: one used for sequential procedures and other for distributed procedures. For sequential procedures, we used a single computer with Intel i7 2.20GHz processor, 16GB of RAM and a 256GB SSD.

For distributed procedures, we set up a Hadoop cluster with a single NameNode and a varying number of DataNodes. This cluster provided both HDFS and MapReduce. All machines had a 2.20GHz Xeon E5-2650 v4 CPU, 8GB of RAM and a 500G SSD. All of them were connected by a 1GBPS network. We used JDK 1.8.0_121, Apache Hadoop 2.7.3 and Apache Jena 3.2.0 for parsing. All these machines were rented at DigitalOcean¹.

6.2 Data

We consider the “truthy” RDF dumps of Wikidata spanning 11 weeks from 2017-04-18 to 2017-06-27. The first version has 1,102,242,331 triples, 54,236,592 unique subjects and 3,276 unique properties, while the final version has 1,293,099,057 triples (+17%), 57,197,406 unique subjects (+5%) and 3,492 unique properties (+6%). Hence we see that the dataset is growing,

¹<https://www.digitalocean.com/>

Table 6.1: Statistics about 11 weeks of truthy RDF dumps provided by Wikidata

Version Date	Size (Gb)	bz2 Size (Gb)	Triples (n)	Subjects (n)	Predicates (n)
2017-04-18	130	4.7	1,102,242,331	54,236,592	3,276
2017-04-25	134	4.8	1,138,975,810	54,433,135	3,305
2017-05-03	134	4.8	1,141,303,957	54,602,559	3,326
2017-05-09	137	4.9	1,170,283,653	54,998,790	3,335
2017-05-16	137	4.9	1,176,093,821	55,127,575	3,364
2017-05-24	138	5.0	1,184,177,503	55,246,566	3,397
2017-05-30	139	5.0	1,190,456,714	55,342,497	3,412
2017-06-07	142	5.1	1,211,813,217	55,449,060	3,448
2017-06-13	145	5.2	1,239,145,151	55,682,498	3,455
2017-06-20	149	5.3	1,271,353,260	56,025,146	3,475
2017-06-27	152	5.4	1,293,099,057	57,197,406	3,492

particularly in the volume of triples (with new triples often using existing properties on existing subjects). Table 6.1 shows simple stats about these data.

6.3 MapReduce Jobs

We code two Mapreduce jobs to extract the characteristics sets. The first one computes the characteristics sets for all subjects, while the second one unifies equal characteristics sets. Figure 6.1 shows a description of these two jobs. We use s , p , and o to refer to *subject*, *predicate*, and *object*: the different parts of an NT-triple.

In the first job, for each NT-triple ($s \ p \ o$) the map task returned a pair (s, p). After the shuffle phase, the reduce task took a stream of properties (p_1, \dots, p_m) for the same s , generated a TreeSet² (a sorted set) with them, and returned a pair ($s, [p_1, \dots, p_m]$), which corresponds to a list of unique properties (characteristic set) per subject.

In the second job, the map tasks took the pair ($s, [p_1, \dots, p_m]$) and returned the characteristic set as a key, and the subject as a value, i.e. ($[p_1, \dots, p_m], s$). After the shuffle phase, the reduce tasks grouped all subjects with the same characteristic set ($([p_1, \dots, p_m], [s_1, \dots, s_n])$).

Summarizing, given an NT file – a representation of an RDF graph – we developed a way to extract its characteristic sets, which will be used to build the lattice from the RDF graph.

²<https://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html>

Task₁ takes as input the set of triples from G and runs:

Map₁ Each input triple (s, p, o) is mapped with key s and value p , thus emitting pairs of the form (s, p) .

Reduce₁ For each key s , the pair $(s, \{p_1, \dots, p_m\})$ is output where $\{p_1, \dots, p_m\}$ is the ordered set of all properties on s .

Task₂ takes as input the set of pairs from **Task₁** and runs:

Map₂ Each input pair $(s, \{p_1, \dots, p_m\})$ is mapped with key $\{p_1, \dots, p_m\}$ and value s .

Reduce₂ For each key $\{p_1, \dots, p_m\}$, all subjects are collected to generate the output $(\{s_1, \dots, s_n\}, \{p_1, \dots, p_m\})$, which corresponds to a CS (and all its associated subjects).

Figure 6.1: Description of MapReduce tasks for computing characteristic sets from an RDF graph G . The letters s , p , and o represent subject, predicate, and object, according to the position in an RDF triple.

6.4 Optimization and Performance Analysis

Initial experiments on the previously described base algorithms encountered memory errors and long runtimes. We thus investigated the following ways to make the process more efficient.

1. We vary the number of reducers in the MapReduce framework.
2. We investigate use of numeric OID compression to reduce disk usage and increase the amount of data loaded in memory.
3. We test internal compression.
4. We code combiner tasks to reduce data sent over the network.

We present performance results for each proposed optimization with the Wikidata RDF dump from 2017-04-18. Rather than trying all combinations (which would take too long to test) we perform a greedy search for the best configuration, where if an optimization proves useful, we use it in all further configurations.

6.4.1 Number of Reducers

To find an efficient configuration of the MapReduce framework, we first perform tests varying the number of reducers for both $Task_1$ and $Task_2$. Table 6.2 shows how the execution time

varies in function of the number of reducers.

We can see that the first task ($Task_1$), which extracts characteristic sets, is considerably slower than the second task ($Task_2$), which groups subjects with the same CS. We can also see that with one reducer we (surprisingly) achieve better performance. While this could be due to several reasons, our best guess is that the size of the key-value pair has much importance.

6.4.2 OIDs representation

In order to reduce disk usage, load more data into memory and reduce execution time, we transformed the NT files into a numeric representation called *Object ID* (or OIDs for short) where IRI strings in the data are replaced with integer identifiers according to a dictionary in a pre-processing step.

Given an NT file, we used one dictionary (key value associative arrays) for subjects and another for predicates. We did not transform objects because they are not necessary for computing characteristic sets.

We scanned the NT file line by line, replacing entities with numeric values. If a subject was not present in the corresponding dictionary, it was added as key with the dictionary's size as value. On the contrary, if a subject was present, we just used its corresponding ID. At the same time we perform the same procedure for the predicates (ignoring the objects).

As a result, we obtained three files that stored: (1) the subject dictionary, (2) the predicate dictionary, and (3) a file that contains the subject and predicate identifiers per line separated by one space. With this transformation, we reduced the size of uncompressed files by a factor of ten. We stored these dictionaries in order to use them in future transformations. For example, when we transformed the file from *20170425* we used the OIDs from *20170418*. This feature guarantees that the same entity will have the same OIDs in all versions of Wikidata.

Before extracting the characteristic sets of the OID file, we updated the first MapReduce job: instead of accepting an NT triple as input, it now receives two numbers (one for subject, the other for predicate). Table 6.3 shows how the execution time varies in function of the

Table 6.2: Execution time of the extraction of characteristic sets with MapReduce from full text input varying the number of reducers.

Job / No. Reducers	1	2	4	8	16	32
$Task_1$	04:09:51	06:22:00	06:19:40	06:20:35	06:16:15	05:35:32
$Task_2$	00:04:09	00:05:27	00:04:16	00:03:09	00:04:50	00:03:39
Total	04:14:00	06:27:27	06:23:56	06:23:44	06:21:05	05:39:11

Table 6.3: Execution time of the extraction of characteristic sets with MapReduce from OIDs representation varying the number of reducers.

Job / No. reducers	1	2	4	8	16	32
Transformation	01:01:32	01:01:32	01:01:32	01:01:32	01:01:32	01:01:32
$Task_1$	00:31:40	00:24:03	00:18:50	00:21:00	00:19:11	00:20:03
$Task_2$	00:01:00	00:01:01	00:00:58	00:01:02	00:01:01	00:01:47
total	01:34:12	01:26:36	01:21:20	01:23:04	01:21:44	01:23:22

number of reducers when we use an OID representation.

Although the OID transformation is a sequential step running on one machine which lasts 01:01:32, comparing Tables 6.2 and 6.3, we can see that the overall execution time is three times smaller when we use the OID representation. Table 6.3 also shows that the best performance is reached with four reducers: one per machine.

6.4.3 Compression

While standard compression techniques can reduce space and transport costs, it can increase CPU times. Hence we run experiments to see if our tasks are better performed with or without standard compression. MapReduce supports two kinds of compression: (1) internal compression, in which all internal processes of MapReduce transparently compress the data written to disk and transferred over the network, and (2) external compression, in which MapReduce accepts as input a bz2 file (among other compression formats).

With previous parameters fixed, we tested the process with both kinds of compression enabled/disabled. Table 6.4 shows how the execution time of characteristic set extraction varies in function of the type of compression used. While using compression can reduce the amount of memory and storage used, it does not improve the performance. Note that external compression of the input file does not affect the second task, which receives output from the first task.

Table 6.4: Execution time of extraction of characteristic sets varying the type of compression.

Job / compression	None	Internal	External
$Task_1$	00:16:39	00:16:45	00:17:23
$Task_2$	00:00:44	00:00:44	—:—:—
total	00:17:23	00:17:29	—:—:—

Table 6.5: Execution time of extraction of characteristic sets in function of the combiners used.

Job / combiner	None	TreeSet	String
$Task_1$	00:16:39	00:14:05	00:17:17
$Task_2$	00:00:44	00:00:53	00:01:07
Total	00:17:23	00:14:58	00:18:24

6.4.4 Combiners

To end our performance analysis, we test if combiners could improve performance. As we covered in Section 2.6.2, a combiner is an optional task between the map task and the shuffle task used to perform a local reduce operation (on the same machine) before the data is sent across the network.

We tested three combinations of combiners and reducers for $Task_1$:

1. No combiner: we used the same reducers described in Section 6.3.
2. Sorted Set concatenation: The combiner adds all properties to a unique sorted set to remove duplicates in an intermediary step and outputs a sorted, unique list of properties.
3. Simple string concatenation: The combiner joins all non-unique properties into a string where the final reducer applies sorting and unification.

The main difference between last two approaches is that the former requires maintaining a sorted list of properties throughout the process, while the latter sorts only at the end. Table 6.5 shows that the second configuration, where a TreeSet is used, has better performance.

6.4.5 Summary

In our final configuration, we use four reducers, OID compression and the sorted set concatenation combiner. With this configuration, we can compute the characteristic sets of the Wikidata dump in *01:16:30* versus *04:14:00* for the original configuration, saving almost 70% of the runtime.

Finally, Table 6.6 shows the total time taken to compute all characteristic sets for all 11 versions of Wikidata over which we will predict changes. We will discuss statistics of these characteristic sets in the following section.

Table 6.6: Execution time of characteristic sets extraction for 11 weeks of Wikidata “truthy” dumps.

Version	Date	OID Transformation	$Task_1$	$Task_2$	Execution time
	2017-04-18	01:01:32	01:10:55	00:08:32	02:20:59
	2017-04-25	01:07:40	01:11:53	00:08:38	02:28:11
	2017-05-03	01:07:30	01:21:05	00:07:32	02:36:07
	2017-05-09	01:09:25	01:16:51	00:08:26	02:34:42
	2017-05-16	01:09:13	01:14:07	00:07:26	02:30:46
	2017-05-24	01:10:54	01:11:20	00:08:38	02:30:52
	2017-05-30	01:11:36	01:15:57	00:08:59	02:36:32
	2017-06-07	01:13:34	01:09:53	00:08:39	02:32:06
	2017-06-13	01:24:57	01:04:24	00:07:45	02:37:06
	2017-06-20	01:18:14	00:58:04	00:06:44	02:23:02
	2017-06-27	01:21:05	01:06:38	00:07:15	02:34:58

6.5 Characteristic Set Statistics

In this section, we are going to present some basic statistics of the extracted characteristic sets in terms of size (number of entities sharing the same characteristic set) and length (number of predicates that are part of a characteristic sets).

Table 6.7 shows the summary of characteristic sets length. We can see that for all versions, the smallest characteristic set is made up of just one property. There are two characteristic sets composed of one property: $\{rdf:type\}$, used to make explicit (one of) the class(es) of an entity, and $\{owl:sameAs\}$, used to say that one entity is the same as another one. We also see that the largest characteristic set increases its size over time, supporting the hypothesis that properties are added to the same entities, making their specifications more complete.

On the other hand, Table 6.8 shows the summary of the size of characteristic sets (the number of entities they have). We can see that most characteristic sets have one entity, while one characteristic sets has almost half of the total entities, which is $\{rdf:type, schema:dateModified, schema:about, schema:version\}$ ³. This is the characteristic set used by Wikidata to represent information about the entities that it stores⁴. The second most popular characteristic set is $\{rdf:type, rdfs:label, skos:prefLabel, schema:name, schema:description, wdt:P31\}$, which corresponds to the minimal information required by Wikidata for an entity to be added to the graph: a type, a label and a description.

In the next chapter, we are going to show how to generate our proposed schema by computing the hierarchy over the extracted characteristic sets.

³schema is the shortcut for <http://schema.org/>

⁴https://www.wikidata.org/wiki/Wikidata:Data_access. Accessed on 28.01.2018

Table 6.7: Summary of characteristic set length (number of properties) for 11 weeks of Wikidata.

Version Date	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
2017-04-18	1	14	18	19.04	22	148
2017-04-25	1	14	18	19.04	22	150
2017-05-03	1	14	18	19.05	23	151
2017-05-09	1	14	18	19.06	23	152
2017-05-16	1	14	18	19.07	23	152
2017-05-24	1	14	18	19.06	23	152
2017-05-30	1	14	18	19.07	23	152
2017-06-07	1	14	18	19.10	23	152
2017-06-13	1	14	18	19.10	23	153
2017-06-20	1	14	18	19.11	23	153
2017-06-27	1	14	18	19.11	23	154

Table 6.8: Summary of characteristic set size (number of entities sharing the same characteristic set).

Version Date	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
2017-04-18	1	1	1	27.05	1	27,096,726
2017-04-25	1	1	1	26.97	1	27,194,839
2017-05-03	1	1	1	26.91	1	27,279,457
2017-05-09	1	1	1	26.98	1	27,477,519
2017-05-16	1	1	1	26.89	1	27,541,728
2017-05-24	1	1	1	26.79	1	27,601,009
2017-05-30	1	1	1	26.70	1	27,648,908
2017-06-07	1	1	1	26.55	1	27,701,941
2017-06-13	1	1	1	26.53	1	27,818,590
2017-06-20	1	1	1	26.56	1	27,989,805
2017-06-27	1	1	1	27.00	1	28,575,838

Chapter 7

Characteristic Set Lattice Computation

Once we have extracted the characteristic sets for 11 versions of Wikidata (see Table 6.6), we are able to structure them in a hierarchical way based on set containment relation (of characteristic sets only) to form a lattice. Remember that we are going to add manually two characteristic sets: the empty set (to be the bottom of the lattice), and the set of all properties (to be the top of the lattice).

In order to compute the characteristic set lattice based on subset relation for n characteristic sets, one approach could be to check $\binom{n}{2}$ (or $\sum_{i=0}^{n-1} i$) pair-wise subset operations. Clearly, this is not practical for *big* values of n . Just consider when $n = 10^5$, the number of function calls is 4,999,950,000. Instead, we adopt the approach outlined in Algorithm 1, which tries to minimize the amount of subset-checking operations.

Let us recall that $\llbracket G \rrbracket$ denotes the set of characteristic sets from an RDF graph G annotated with their sets of subjects. We say that $(S, P) \in \llbracket G \rrbracket$ if $S \subseteq \pi_s(G)$, $P \subseteq \pi_p(G)$, and S the set of entities having the characteristic set P (See Remark 2.3). Let us also recall that $\llbracket G \rrbracket^*$ is an extension of $\llbracket G \rrbracket$ where we add manually the top and the bottom characteristic sets (See Section 4.2). Here we denote the set of characteristic sets by \mathcal{P} , i.e., $\mathcal{P} = \{P \mid \exists S : (S, P) \in \llbracket G \rrbracket^*\}$.

The first step of Algorithm 1 is to stratify these characteristics sets by the number of properties that they contain, generating the notion of “levels”. We say that $\mathcal{P}.n = \{P \in \mathcal{P} : |P| = n\}$. We then proceed to compare the characteristic sets between levels looking for direct subset containments. To do so, we nest one reversed loop into a normal loop starting from 2, i.e., we start with $j = \mathcal{P}.2$ to $j = \mathcal{P}.m$ (where m is the maximum number of levels) comparing the levels from $i = \mathcal{P}.j - 1$ to $i = \mathcal{P}.1$. In other words, we compare levels in the order $(\mathcal{P}.2, \mathcal{P}.1), (\mathcal{P}.3, \mathcal{P}.2), (\mathcal{P}.3, \mathcal{P}.1), \dots, (\mathcal{P}.m, \mathcal{P}.1)$.

Table 7.1: Statistics about lattices for 11 weeks of Wikidata RDF Graphs

Version Date	Execution Time	No. nodes	No. edges
2017-04-18	06:57:59	2,004,910	78,046,423
2017-04-25	07:06:31	2,018,129	79,010,989
2017-05-03	07:07:54	2,028,664	79,621,809
2017-05-09	07:27:06	2,037,955	80,145,091
2017-05-16	07:20:06	2,049,553	81,172,618
2017-05-24	07:21:17	2,061,984	81,971,396
2017-05-30	07:29:35	2,072,240	82,753,478
2017-06-07	07:38:51	2,088,388	84,554,441
2017-06-13	07:39:08	2,098,769	85,347,491
2017-06-20	07:47:32	2,109,063	86,168,870
2017-06-27	07:51:07	2,118,109	86,848,506
all		2,553,486	93,613,776

We have two algorithms to compare the characteristic sets from $\mathcal{P}.i$ and $\mathcal{P}.j$ ($i < j$): REMOVEONE and RAREJOIN.

removeOne: When $i + 1 = j$, we invoke REMOVEONE, where from each characteristic set in $\mathcal{P}.j$, we remove one property at a time and check if the result was in $\mathcal{P}.i$. We check empirically that looking up $|\mathcal{P}.j| \times j = j^2$ times for a characteristic set in $\mathcal{P}.i$ was faster than checking $|\mathcal{P}.j| \times |\mathcal{P}.i| = i \times j$ times if a characteristic set was subset of another. The key of this process is to use an index to check membership in $\mathcal{P}.i$, which is $\mathcal{O}(1)$, while the subset operation is $\mathcal{O}(i)$.

rareJoin: Otherwise we apply RAREJOIN where, for each characteristic set $P_i \in \mathcal{P}.i$, we find the rarest property $p \in P_i$ in terms of appearing in the fewest sets of \mathcal{P} (choosing arbitrarily based on lexical order if tied), retrieve each $P_j \in \mathcal{P}.j$ that also contains p , and then check if $P_i \subset P_j$ and (P_i, P_j) is not already reachable in the current partial order; if so, we add the pair (P_i, P_j) to the partial order. Note that in a preprocessing step, all properties in each input characteristic set are ordered by rarest first, and we create an inverted index from properties to characteristic sets by level; hence finding all P_j matching the condition on Line 25 requires one lookup on the inverted index. While the upper-bound remains $|\mathcal{P}.i| \times |\mathcal{P}.j|$ set-containment checks, in practice, comparing only pairs of sets that share their rarest property should greatly reduce the number of comparisons from a brute-force method.

In terms of the condition for choosing one algorithm or the other, note that if we considered a generalized method REMOVE N for $n = j - i \leq N$, we would end up having to perform $\binom{j}{n}$ lookups on the $\mathcal{P}.i$ index, which would be problematic for $n \approx \frac{j}{2}$. Empirically we found that REMOVEONE was the only case faster than RAREJOIN.

We finally agglomerate all direct subsets into a list, and then return it. To guarantee that we only included direct subset relations, we maintain in memory the previous direct

subsets and check if there was a path in the lattice between the node to be compared (next characteristic set in the sequence) and any other characteristic set that has an edge to the adding node (characteristic set being compared).

Table 7.1 show statistics about the lattice computation for the 11 weeks of Wikidata. We can see that the execution time is linear except for version *2017-05-09*. Our best guess is that at the time another process was executed on the machine. We remark that the number of nodes corresponds to the number of characteristic sets per version.

Note that both the diff operator (see Section 5.1) and the add operator (see Section 5.2) are defined over the union of characteristic sets, i.e. $\mathcal{P}_i \cup \mathcal{P}_j$, hence we just need to compute one lattice over C instead of two lattices. This feature can be easily extended to n lattices. That is why in Table 7.1 the last row (version *all*) appears.

Algorithm 1 Computing the characteristic set lattice ($\llbracket G \rrbracket^*, \subseteq$)

```

1: function POSET( $\llbracket G \rrbracket^*$ )
2:    $\mathcal{P} \leftarrow \{P \mid \exists S : (S, P) \in C\}$  ▷ we only need the characteristic sets
3:   initialize  $\ominus$  ▷ will store the direct containments:  $\ominus \subseteq \mathcal{P} \times \mathcal{P}$ 
4:   let  $\mathcal{P}.n := \{P \in \mathcal{P} : |P| = n\}$  ▷ returns characteristic sets on level  $n$ 
5:    $m \leftarrow \max\{|P| : P \in \mathcal{P}\}$  ▷ the size of the largest characteristic set
6:   for  $j = 2; j \leq m; j++$  do
7:     for  $i = j - 1; i > 0; i--$  do
8:       if  $i = j - 1$  then
9:          $\ominus \leftarrow \ominus \cup \text{REMOVEONE}(\mathcal{P}.i, \mathcal{P}.j)$ 
10:      else
11:         $\ominus \leftarrow \ominus \cup \text{RAREJOIN}(\mathcal{P}.i, \mathcal{P}.j, \ominus)$ 
12:   let  $(S, P) \prec (S', P')$  if and only if  $(P, P') \in \ominus$ 
13:   return  $(C, \prec)$ 

14: function REMOVEONE( $\mathcal{P}_i, \mathcal{P}_j$ ) ▷ for  $P_j \in \mathcal{P}_j$ , remove  $p \in P_j$ , see if set in  $\mathcal{P}_i$ 
15:   initialize  $\ominus_{i,j}$ 
16:   for  $P_j \in \mathcal{P}_j$  do
17:     for  $p \in P_j$  do
18:        $P'_j \leftarrow P_j \setminus \{p\}$ 
19:       if  $P'_j \in \mathcal{P}_i$  then
20:          $\ominus_{i,j} \leftarrow \ominus_{i,j} \cup \{(P_i, P_j)\}$ 
21:   return  $\ominus_{i,j}$ 

22: function RAREJOIN( $\mathcal{P}_i, \mathcal{P}_j, \ominus$ ) ▷ check  $\subset$  only for sets sharing rare property
23:   initialize  $\ominus_{i,j}$ 
24:   for  $P_i \in \mathcal{P}_i$  do
25:     for  $P_j \in \mathcal{P}_j : P_i[0] \in P_j$  do ▷ for all  $P_j$  containing rarest prop. of  $P_i$ 
26:       if  $P_i \subset P_j \wedge (P_i, P_j) \notin \ominus^+$  then ▷ if subset and not reachable in  $\ominus$ 
27:          $\ominus_{i,j} \leftarrow \ominus_{i,j} \cup \{(P_i, P_j)\}$ 
28:   return  $\ominus_{i,j}$ 

```

Chapter 8

High Level Dynamics Evaluation

In previous chapters, we had proposed a hierarchical schema for RDF graphs based on formal concepts analysis (see Chapter 4), and we had proposed algebraic methods over such schemas to model the dynamics of RDF graphs (see Chapter 5). On the other hand, we had extracted the characteristic sets of 11 versions of Wikidata (see Chapter 6), and we had computed the Characteristic Set #-Lattice – our schema proposal – of those versions (see Chapter 7).

In this chapter, we are going to evaluate our schema proposal over a concrete use case: predicting how the hierarchical schema evolves over time.

The idea behind our experiment is to train on w previous weekly versions of the dataset to predict the next version of the Characteristic Set #-Lattice. Given that we have 11 versions, we train on $1 \leq w \leq 6$ versions to ensure at least 5 ($11 - w$) predicted lattices for each experiment.

We are going to use (1) our algebraic method, and (2) a Linear Regression Model (LM) as a baseline to predict the next schema. To use our algebraic method, we averaged the w previous diffs and added it to the latest version. To use a LM, we consider the number of entities in each characteristic set as an independent variable that changes over time.

To measure the quality of the prediction, we compute the RMSE (see Equation 2.3) and the MAE (see Equation 2.4) between the predicted Characteristic Set #-Lattice and the real lattice. To do so, we ordered the nodes of both predicted and real lattices (storing its number of subjects) in an one dimensional vector.

Over this configuration, we performed two experiments: a *flat* prediction, where we considered the exact number of subjects per characteristic set, and a *transitive* prediction, where we added up all transitive subsets of a characteristic set.

Table 8.1: RMSE and MAE of predicted Characteristic Set #-Lattice for exact characteristic set.

w	LM (RMSE)	LM (MAE)	Δ (RMSE)	Δ (MAE)
2	167.2	0.5697	25.26	0.1286
3	173.9	0.5595	19.15	0.1134
4	186.6	0.6051	17.71	0.1078
5	196.0	0.6624	17.31	0.1020
6	202.9	0.6842	15.62	0.0941

8.1 Flat Prediction

The first experiment considers the counts of subjects with an exact characteristic set, evaluating the quality of prediction given an exact characteristic set; this is useful, for example, to predict how the characteristic set of a particular subject might change in future.

The results are shown in Table 8.1, where we see that our diff algebra (Δ) outperforms the baseline method (LM) in all cases, with smaller error by a considerable margin. In particular, when we use six weeks to predict the next one, our method obtains an RMSE 12 times smaller than the linear regression model, while the MAE was 7 times smaller.

We attribute this to the fact that Δ considers where entities come from, whereas LM does not, i.e., LM does not care if there are no entities in a characteristic set to “move” to another, nor where the entities are going to move.

We also see that considering more weeks improves the quality of prediction for Δ : considering further training data allows the model to smooth out the effect of certain bursty (e.g., bot) edits between recent versions.

8.2 Transitive Prediction

The second experiment we run considers the counts of subjects with at least a given characteristic set, but that may have further properties. A concrete use-case for this situation would be to predict how the results for a query with those properties may change.

Table 8.2 shows these results. First, we note that the overall error rises considerably, which is to be expected as the absolute (transitive) counts likewise increase considerably.

As before, we see that the Δ -based predictions considerably outperform the LM baseline, and that the errors decrease for Δ as further weeks of training data are considered for the prediction. Analogously to the previous computation, when we used six weeks to predict the next week, our method obtains an RMSE 9 times smaller than the linear regression model,

Table 8.2: RMSE and MAE of predicted Characteristic Set #-Lattice for transitive subset characteristic set

w	LM (RMSE)	LM (MAE)	Δ (RMSE)	Δ (MAE)
2	1477.8	177.0	264.2	6.19
3	1458.9	162.4	209.1	5.09
4	1535.6	178.8	185.7	4.50
5	1398.8	123.4	176.7	4.15
6	1357.8	59.6	145.8	3.67

while the MAE was 16 times smaller.

Please note that data, source code and other evaluation materials are available at:
https://github.com/larryjgonzalez/rdf_dynamics.

Chapter 9

Conclusions

In this thesis, we have proposed a new hierarchical data-driven schema for large-scale graphs based on formal concept analysis. Because several authors have reported difficulties when computing the formal concepts, we propose to avoid computing the formal concepts and use the characteristic sets instead. With them, we build a Characteristic Set Lattice, which is our resulting hierarchical schema proposal.

We discussed algorithms for computing these characteristic sets in an scalable way, and build the lattice in an efficient manner. We also presented a performance analysis about the characteristic set extraction.

We not only presented a hierarchical data-driven schema for graph databases, but also algebraic methods to generate a diff between such schemas, and to add a diff to a given schema. With these algebraic operations, we were able to focus on a concrete use-case: to predict high-level changes in a graph database.

We chose Wikidata as the data source for our evaluation. We compute the characteristic sets of 11 versions of Wikidata – each with more than 1 billion triples, 50 million subjects and 3 thousand properties – and predicted how their schemas will evolve. Finally, we validated the quality of predictions made by our algebraic approach against a simple linear regression model baseline.

Because of all these mentioned reasons, we validate our research hypothesis (see Section 1.1), i.e., we were able to define and compute a hierarchical schema from an RDF using its characteristic sets, and we were able to use such schema to model the dynamics of those graphs.

In terms of future directions, there are several ways to continue and improve this work. While we focus on computing the characteristics sets as an initial proposal, there are some alternatives to explore: (1) We could compute the characteristics sets from the objects, instead of the subjects and consider it as a *inverse* lattice. (2) We could combine the normal lattice and the inverted lattice. (3) We could improve the characteristics sets counting the number of times a predicate appears. Other variations of schema could also be explored,

including, for example, concepts that encode type values or multiplicity, or quotient graphs based on characteristic sets.

Without modifying the structure of the characteristic sets, a natural step could be to use more data. Currently, we have more than nine months of Wikidata dumps; however, we don't have dumps for all weeks. In the same line, we could add different data sources, like *bio2RDF*¹, *DBpedia*², *IMGpedia*³, among others. These datasets could be used to validate our proposals at different scales and over datasets with different types of dynamic behavior.

Improving efficiency is always a possible future step. While some of our algorithms were optimized for memory or time, others were not. In this case, we could perform a fine-grained analysis of algorithms, or we could use Compact Data Structures[49], to load more data into memory and reduce the execution time.

While we use a linear model to compare our method for predicting how the proposed schema would evolve, several other techniques could be used: Time Series Analysis [40], Artificial Networks [57], Support Vector Machine [60], etc. Furthermore, we could apply transformations to our training data – the number of subjects per characteristic set – to search for better *RMSE* and *MAE*.

On the other hand, we could reduce the number of characteristic sets by merging them. While this would require an extra step, it could reduce the amount of time spent to compute the lattice, and it could facilitate its understanding.

Leaving aside the improvements to the core algorithms, we could use the schema for different applications: We could create an application where users could navigate the schema; we could test if our schema is helpful to formulate SPARQL queries or to choose an optimal plan for query processing; we could improve faceted navigation, graph databases, and so on and so forth.

A much more ambitious alternative – although we are not certain about the technical feasibility – is to compute the actual *concepts* and *concept lattice* (see Section 2.2) and use it as a hierarchical schema.

In general, we foresee much potential in the area of deriving data-driven schema from graphs.

¹<http://bio2rdf.org/>

²<http://wiki.dbpedia.org/>

³<http://imgpedia.dcc.uchile.cl/>

Bibliography

- [1] Apache hadoop. <http://hadoop.apache.org/>. Accessed: 2018-01-07, Last Published: i2017-12-18.
- [2] Hadoop mapreduce. https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html. Accessed: 2018-01-07, Last Published: 2013-08-04.
- [3] Rdf dump format. https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format. Accessed: 2018-02-13.
- [4] Abedjan, Z., Gruetze, T., Jentzsch, A., and Naumann, F. (2014). Profiling and mining rdf data with prolod++. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 1198–1201. IEEE.
- [5] Abedjan, Z. and Naumann, F. (2013). Improving rdf data through association rule mining. *Datenbank-Spektrum*, 13(2):111–120.
- [6] Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc.
- [7] Alam, M., Buzmakov, A., Codocedo, V., and Napoli, A. (2015). Mining definitions from rdf annotations using formal concept analysis. In *IJCAI*, pages 823–829.
- [8] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer.
- [9] Ayers, P., Matthews, C., and Yates, B. (2008). *How Wikipedia works: And how you can be a part of it*. No Starch Press.
- [10] Bechhofer, S. (2009). Owl: Web ontology language. In *Encyclopedia of database systems*, pages 2008–2009. Springer.
- [11] Beckett, D., Berners-Lee, T., Prud’hommeaux, E., and Carothers, G. (2014). Rdf 1.1 turtle. *World Wide Web Consortium*.
- [12] Berners-Lee, T., Hendler, J., Lassila, O., et al. (2001). The semantic web. *Scientific american*, 284(5):28–37.
- [13] Bordat, J.-P. (1986). Calcul pratique du treillis de galois d’une correspondance. *Mathématiques et Sciences humaines*, 96:31–47.
- [14] Borthakur, D. et al. (2008). Hdfs architecture guide. *Hadoop Apache Project*, 53.

- [15] Brickley, D., Guha, R. V., and McBride, B. (2014). Rdf schema 1.1. *W3C recommendation*, 25:2004–2014.
- [16] Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117.
- [17] Campinas, S., Perry, T. E., Ceccarelli, D., Delbru, R., and Tummarello, G. (2012). Introducing rdf graph summary with application to assisted sparql formulation. In *Database and Expert Systems Applications (DEXA), 2012 23rd International Workshop on*, pages 261–266. IEEE.
- [18] Čebirić, Š., Goasdoué, F., and Manolescu, I. (2015). Query-oriented summarization of rdf graphs. *Proceedings of the VLDB Endowment*, 8(12):2012–2015.
- [19] Čebirić, Š., Goasdoué, F., and Manolescu, I. (2017). *Query-oriented summarization of RDF graphs*. PhD thesis, INRIA Saclay; Université Rennes 1.
- [20] Chein, M. (1969). Algorithme de recherche des sous-matrices premières d’une matrice. *Bulletin mathématique de la Société des Sciences Mathématiques de la République Socialiste de Roumanie*, pages 21–25.
- [21] Cimiano, P., Hotho, A., and Staab, S. (2005). Learning concept hierarchies from text corpora using formal concept analysis. *J. Artif. Intell. Res.(JAIR)*, 24(1):305–339.
- [22] Consortium, W. W. W. et al. (2014a). Rdf 1.1 concepts and abstract syntax.
- [23] Consortium, W. W. W. et al. (2014b). Rdf 1.1 primer.
- [24] d’Aquin, M. and Motta, E. (2011). Extracting relevant questions to an rdf dataset using formal concept analysis. In *Proceedings of the sixth international conference on Knowledge capture*, pages 121–128. ACM.
- [25] Dau, F. and Sertkaya, B. (2011). Formal concept analysis for qualitative data analysis over triple stores. In *International Conference on Conceptual Modeling*, pages 45–54. Springer.
- [26] Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.
- [27] Dividino, R., Gottron, T., and Scherp, A. (2015). Strategies for efficiently keeping local linked open data caches up-to-date. In *International Semantic Web Conference*, pages 356–373. Springer.
- [28] Dürst, M. and Suignard, M. (2004). Internationalized resource identifiers (iris). Technical report.
- [29] Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.-H., Qiu, J., and Fox, G. (2010). Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM international symposium on high performance distributed computing*, pages 810–818. ACM.
- [30] Ellison, N. B. et al. (2007). Social network sites: Definition, history, and scholarship. *Journal of computer-mediated Communication*, 13(1):210–230.

- [31] Fernández, J. D., Polleres, A., and Umbrich, J. (2015). Towards efficient archiving of dynamic linked open data. In *DIACRON@ ESWC*, pages 34–49.
- [32] Formica, A. (2012). Semantic web search based on rough sets and fuzzy formal concept analysis. *Knowledge-Based Systems*, 26:40–47.
- [33] Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The elements of statistical learning*, volume 1. Springer series in statistics New York.
- [34] Fu, H. and Nguifo, E. M. (2004). A parallel algorithm to generate formal concepts for large data. In *International Conference on Formal Concept Analysis*, pages 394–401. Springer.
- [35] Ganter, B. (2010). Two basic algorithms in concept analysis. In *International conference on formal concept analysis*, pages 312–340. Springer.
- [36] González, L. and Hogan, A. (2017). A data-driven graph schema. In *AMW 2018 12th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Cali, Colombia, May 23-25, 2018*. Barbara Pobleto, Dan Olteanu.
- [37] González, L. and Hogan, A. (2018). Modelling dynamics in semantic web knowledge graphs with formal concept analysis. In *Proceedings of the 2018 World Wide Web Conference, WWW '18*, pages 1175–1184, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.
- [38] Grant, J. and Beckett, D. (2004). Rdf test cases. *W3C recommendation*, 10.
- [39] Gutierrez, C., Hurtado, C. A., Mendelzon, A. O., and Pérez, J. (2011). Foundations of semantic web databases. *Journal of Computer and System Sciences*, 77(3):520–541.
- [40] Hamilton, J. D. (1994). *Time series analysis*, volume 2. Princeton university press Princeton.
- [41] Käfer, T., Abdelrahman, A., Umbrich, J., O’Byrne, P., and Hogan, A. (2013). Observing linked data dynamics. In *Extended Semantic Web Conference*, pages 213–227. Springer.
- [42] Kaufmann, E. (2008). *Talking to the semantic web: natural language query interfaces for casual end-users*. PhD thesis, University.
- [43] Kinsella, S., Bojars, U., Harth, A., Breslin, J. G., and Decker, S. (2008). An interactive map of semantic web ontology usage. In *Information Visualisation, 2008. IV’08. 12th International Conference*, pages 179–184. IEEE.
- [44] Kirchberg, M., Leonardi, E., Tan, Y. S., Link, S., Ko, R. K., and Lee, B. S. (2012). Formal concept discovery in semantic web data. In *International Conference on Formal Concept Analysis*, pages 164–179. Springer.
- [45] Krajca, P., Outrata, J., and Vychodil, V. (2008). Parallel recursive algorithm for fca. In *CLA*, volume 2008, pages 71–82. Citeseer.

- [46] Krajca, P. and Vychodil, V. (2009). Distributed algorithm for computing formal concepts using map-reduce framework. In *International Symposium on Intelligent Data Analysis*, pages 333–344. Springer.
- [47] Krötzsch, M., Vrandečić, D., Völkel, M., Haller, H., and Studer, R. (2007). Semantic wikipedia. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(4):251–261.
- [48] Meusel, R., Petrovski, P., and Bizer, C. (2014). The webdatacommons microdata, rdfa and microformat dataset series. In *International Semantic Web Conference*, pages 277–292. Springer.
- [49] Navarro, G. (2016). *Compact data structures: A practical approach*. Cambridge University Press.
- [50] Neumann, T. and Moerkotte, G. (2011). Characteristic sets: Accurate cardinality estimation for rdf queries with multiple joins. In *2011 IEEE 27th International Conference on Data Engineering*, pages 984–994. IEEE.
- [51] Norris, E. M. (1978). An algorithm for computing the maximal rectangles in a binary relation. *Revue Roumaine de Mathématiques Pures et Appliquées*, 23(2):243–250.
- [52] Nourine, L. and Raynaud, O. (1999). A fast algorithm for building lattices. *Information processing letters*, 71(5-6):199–204.
- [53] Pham, M.-D. and Boncz, P. (2016). Exploiting emergent schemas to make rdf systems more efficient. In *International Semantic Web Conference*, pages 463–479. Springer.
- [54] Pham, M.-D., Passing, L., Erling, O., and Boncz, P. (2015). Deriving an emergent relational schema from rdf data. In *Proceedings of the 24th International Conference on World Wide Web*, pages 864–874. International World Wide Web Conferences Steering Committee.
- [55] Picalausa, F., Luo, Y., Fletcher, G. H., Hidders, J., and Vansummeren, S. (2012). A structural approach to indexing triples. In *Extended Semantic Web Conference*, pages 406–421. Springer.
- [56] Prud, E., Seaborne, A., et al. (2006). Sparql query language for rdf.
- [57] Schalkoff, R. J. (1997). *Artificial neural networks*, volume 1. McGraw-Hill New York.
- [58] Shadbolt, N., Berners-Lee, T., and Hall, W. (2006). The semantic web revisited. *IEEE intelligent systems*, 21(3):96–101.
- [59] Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., and Reynolds, D. (2008). Sparql basic graph pattern optimization using selectivity estimation. In *Proceedings of the 17th international conference on World Wide Web*, pages 595–604. ACM.
- [60] Suykens, J. A. and Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300.

- [61] Umbrich, J., Decker, S., Hausenblas, M., Polleres, A., and Hogan, A. (2010). Towards dataset dynamics: Change frequency of linked open data sources.
- [62] Wille, R. (1982). Restructuring lattice theory: an approach based on hierarchies of concepts. In *Ordered sets*, pages 445–470. Springer.
- [63] Xu, B., de Fréin, R., Robson, E., and Foghlú, M. Ó. (2012). Distributed formal concept analysis algorithms based on an iterative mapreduce framework. In *International Conference on Formal Concept Analysis*, pages 292–308. Springer.

List of Figures

2.1	Hasse diagram	9
2.2	Movie stars with their attributes	11
2.3	Concept lattice	12
2.4	Two RDF triples with full IRIs and prefix shortcuts	13
2.5	RDF graph example in RDF	14
2.6	Graph representation of RDF graph	15
2.7	Characteristic sets annotated with entities	16
2.8	Robert De Niro’s Wikidata webpage	18
4.1	RDF example in Turtle Syntax.	28
4.2	Set of entiteis and propertis from RDF graph	28
4.3	Cross table representing the formal context	29
4.4	Formal concepts	29
4.5	Hasse diagram of formal context	29
4.6	Characteristic sets annotated with entities ($\llbracket G \rrbracket^*$)	30
4.7	Hasse diagram of the Characteristic Set Lattice	31
4.8	Characteristic sets annotated with the number of subjects $\llbracket G \rrbracket^\#$	31
4.9	Hasse diagram of the Characteristic Set $\#$ -Lattice	32
5.1	Diff $\Delta_{2,1}$ of two Characteristic Set Lattice	34
5.2	Cardinality diff of two Characteristic Set Lattices	35

5.3	Add a diff to an Characteristic Set #-Lattice	37
6.1	Description of MapReduce tasks	40

List of Tables

2.1	Namespace prefix and IRIs	14
6.1	Statistics about 11 weeks of Wikidata dumps	39
6.2	Execution time of characteristic sets extraction using full text input	41
6.3	Execution time of characteristic sets extraction using OIDs representation	42
6.4	Execution time of characteristic sets extraction using compression	42
6.5	Execution time of characteristic sets extraction using combiners	43
6.6	Execution time of characteristic sets extraction for 11 weeks of Wikidata	44
6.7	Summary of characteristic set length	45
6.8	Summary of characteristic set size	45
7.1	Statistics about lattices for 11 weeks of Wikidata RDF Graphs	47
8.1	Error for exact Characteristic Set #-Lattice prediction	50
8.2	Error for transitive Characteristic Set #-Lattice prediction	51