UniBa UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

Università degli Studi di Bari Aldo Moro

Computer Science Department

PhD Programme in Computer Science and Mathematics

XXXVII Cycle

Scientific Disciplinary Area INFO/01

Ontology-enriched Graph Databases: an Interoperable Framework for Knowledge Integration and Management

Coordinator:
Prof. Francesca MAZZIA

PhD Candidate:
Davide DI PIERRO

Tutor:
Prof. Stefano FERILLI

Co-Tutor:
Dr. Domenico REDAVID

FINAL EXAM 2025

Abstract

The amount and variety of data available nowadays are constantly increasing due to academic and industrial needs, as well as for building intelligent and sustainable network systems for society. Researchers endeavour to keep pace with this expansion with new hardware and software solutions, but they also struggle to design solutions that are compliant with new regulations, environmentally sustainable, and manageable in the long term. Given the extraordinary quantity of domains in which data are employed, the natural consequence is they come under diverse formats and with different vocabularies. In this context, graph databases are extremely valuable given their unfixed structure, which allows flexibility and permits facilitating time-consuming tasks like data integration. The cause of this flexibility is that graph databases do not hold schema information about instances, which is in contrast with traditional database logical models. The natural downside of this peculiarity is the lack of any control over the available information. In this scenario, problems related to data disambiguation, incompleteness or inconsistency are frequent. Here we propose the enrichment of graphs with schema information, combining in a single solution the advantages of graphs in terms of performance and interoperability and the aspects of sharing a common vocabulary that, seen as an ontological conceptualization, opens scenarios for reasoning capabilities.

The contribution of the thesis is the design of a framework for integrating schemas into graph-based structures. Schemas solve or mitigate the above-mentioned issues in data integration, and help in solving current Artificial Intelligence tasks involving graph structures efficiently. The contribution includes a solution for interconnecting and mapping graph database models and the SW-based graph model. The proposed framework covers a solution for schema evaluation that is purely instance-based and demonstrates the capability of this solution to be integrated into first-order logic frameworks for reasoning tasks. The proposed framework is a general solution and, among its possible uses, we mainly experimented with the framework in the field of digital libraries, cultural heritage and linguistics.

Keywords: graph databases, ontology, knowledge representation, artificial intelligence, semantic web.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

ASP Answer Set Programming

CWA Closed World Assumption

DB Database

DBMS DataBase Management System

DL Description Logics

GBS GraphBRAIN Schema

IRI Internationalized Resource Identifier

JSON JavaScript Object Notation

KB Knowledge Base

KG Knowledge Graph

KRR Knowledge Representation and Reasoning

LOD Linked Open Data

LP Logic Programming

LPG Labeled Property Graph

ML Machine Learning

NLP Natural Language Processing

OO Object-Oriented

OODBMS Object-Oriented DBMS

ORDBMS Object-Relational DBMS

OWA Open World Assumption

OWL Ontology Web Language

PG Property Graph

QL Query Language

RDF Resource Description Framework

RDFS RDF Schema

SPARQL SPARQL Protocol and RDF Query Language

SPG Singleton Property Graph

SQL Structured Query Language

SW Semantic Web

URI Uniform Resource Identifier

WWW World Wide Web

XML eXtensible Markup Language

Chapter 1

Introduction

The amount of data available keeps increasing every day. This continuous flow comes from devices, sensors, IoT systems, databases and other possible sources. The problem of merging data has been taken into account since the beginning of the discipline; however, it has become extremely relevant in the modern realm of data. With the advent of many database logical models to store data, technical difficulties in combining heterogeneous sources multiply. Based on the quantity, variety, and use of these data, many logical models have been implemented to store information, but the integration among them is still unsatisfactory. In this context, graphs play a central role given their interpretative representation of instance relationships. In this contribution, I will mainly refer to the Labeled Property Graph (LPG) model, which is the logical model supported by Neo4j[1]. This technology facilitates this integration as the structure is not fixed as in previous data modelling approaches. The main peculiarity of this NoSQL model [1] is the lack of a schema, which makes consistency and disambiguation open challenges. Many solutions have been employed so far to address this issue, but no general agreement in the community has been reached. Almost every logical model has been mapped onto a graph-based model. This has been the result of the spread and research challenges graphs pose.

In the field of Artificial Intelligence (AI), graphs have been playing a fundamental role since the beginning of the discipline. Specifically, graph theory influenced automata and type theory for verification and model checking. It has been used to formalize process mining settings and now represents the basic structure for represent-

[1] https://neo4j.com/

ing other symbolic representations. The field of AI comprises two macro-categories of techniques that differentiate from the task to be solved, the adopted algorithms, and the results, but also the reliability and some considerations vary. We refer to *symbolic* approaches those relying mostly on logic, symbol manipulation and logical reasoning. They were the first AI approaches to be introduced but are still in vogue today. Much attention is being paid today to these techniques because they can support explainability, which is becoming increasingly fundamental, especially in some critical application domains such as recommendation systems, medicinal tools, tutoring systems and so on. In contrast, *subsymbolic* approaches rely on relevant statistical analysis to uncover meaningful patterns and relations among data. The former are inspired by human reasoning and, hence, tend to be explainable and interpretable for final users (experts or not). They are not, in general, difficult to extend but tend to be domain-specific and poorly scalable. Conversely, the latter are now much more complex from a computational point of view, they scale better and suffer less from the specificity of the domain. In almost all cases, the results lack explainability, making the output interpretation process demanding. The two categories can be referred to as complementary since the advantages of the first are the disadvantages of the latter and vice versa. Yet there is room for trying to take the best of both worlds, and graphs are pioneers for this. Graphs are a full-fledged storage representation formalism for strongly related information. Many *subsymbolic* techniques, mainly based on statistical reasoning, have been adopted for knowledge completion tasks like link prediction [2] or node attribute inference [3], but also for summarization (e.g. graph coarsening [4]) or ranking (PageRank [5]). On the other hand, logical reasoning is prone to be used on graphs given the interpretability of relationships to which, in general, a semantic is attached. Among the first uses of graphs we mention explainable recommender systems [6, 7] or critical systems [8].

When dealing with knowledge integration, it is worth mentioning the Semantic Web (SW), born as a sharing metadata technology for knowledge on the Web. In this context, Knowledge Representation and Reasoning (KRR) and SW communities proposed the Resource Description Framework (RDF) and RDF Schema to describe both the schemas (in the form of ontologies) and the data, in the form of triples. Thus, the need arises to integrate tools for representing databases with tools for KRR and there is room for research on their integration. In this context, the LPG and RDF

models are different and partly incompatible, but the fusion would yield more value. The reconciliation or integration of the two graph models has great potential but the solutions proposed so far are still unsatisfactory mainly due to a lack of generality. The integration opens scenarios and possibilities for users. Users might be interested in integrating into a consistent graph database community knowledge taken from the "open" world.

Apart from ontological reasoning, logic frameworks can also express graph-based structures. Reasoning about graphs in logic frameworks has been an underrepresented line of research but some potentialities have somehow been discussed. Specifically, critical operations may be translated as graph-based logic programs thanks to the above-mentioned positive properties of this setting.

When dealing with ontological conceptualizations, the problem of how to evaluate the *quality* of the schemas arises. Different levels of detail positively or negatively affect the quality and performance of schema management and the reasoning processes derived from them.

1.1 Research Questions

The study aims to address the following research questions:

1. How can we introduce schemas on graph databases to solve modern AI problems?
2. How can we map this setting with existing semantics-based formalisms?
3. How can we evaluate graph schemas?
4. How can we reason with graphs in a Logic Programming fashion?

1.2 Contribution of This Work

The main contribution of this work was the development of a new framework for creating, storing, managing, interconnecting and reasoning with graph-based data. The framework allows the creation and management of graph schemas, which govern the structures and constraints of the graph instances. Schemas are compliant with a subset of standard ontological concepts but also introduce nonstandard novel constraints.

Given the introduction of schemas, I provided a strategy to map schemas and data onto the standard concepts available in the SW community. The integration is aimed at exporting resources, making use of the public knowledge available, and preserving (and possibly extending) information (e.g., disambiguating classes and properties). Introduced schemas can also be evaluated with respect to graph instances, to evaluate whether they are overly complex or new refinements can be introduced. The schema evaluation and the refinement suggestion process are instance-based and of support for a schema designer and/or domain expert. Finally, the contribution aimed to introduce logic frameworks for reasoning with graph and graph schemas, specifically for tasks involving the communication between the two. The framework has been adopted in several use domains, demonstrating its usefulness and generality. The framework can naturally be adopted with available datasets, schemas or a combination of both.

1.3 Structure of the Thesis

This thesis is organized as follows:

- Chapter 1 introduces the topic, the motivations, the research questions of the thesis and the contribution of this work.
- Chapter 2 provides background knowledge about technologies and fields related to this work.
- Chapter 3 describes the state of the art in the different fields in which this work resides.
- Chapter 4 describes the system I used as a starting point and on which I implemented the solution.
- Chapter 5 describes the proposed methodologies for solving AI problems in different domains.
- Chapter 6 illustrates the mapping of the solution with existing SW standard, proposing a semi-automatic mapping with personalization.
- Chapter 7 describes an instance-based evaluation process of the proposed schemas.

- Chapter 8 describes several logic programming-based solutions within the context of the proposed methodology.
- Chapter 9 summarizes the work, describes possible consequences, highlights limitations and extensions, and concludes the manuscript.

Chapter 2

Background

In this chapter, I laid the foundations for the main elements involved in the project. This was not intended to be a comprehensive survey but only to describe the ingredients, the reasons why we are interested and the main (current or historical) challenges.

2.1 Ontologies

The term *ontology* has origins in philosophy. The Computer Science community understood the need to catalogue and classify a domain since the early beginning of the discipline. The word `ontology` is used with different meanings in different communities. It can be distinguished between the use as an uncountable noun (`Ontology`, with uppercase initial) and the use as a countable noun (`an ontology`, with lowercase initial). In the first case, it is referred to as a philosophical discipline, namely the branch of philosophy which deals with the nature and structure of reality. In the second case, which reflects the most prevalent use in Computer Science, it is referred to an `ontology` as a special kind of information object or computational artefact [9]. In both cases, it can be referred to as the process of identifying all the kinds and structures of objects, properties, events, processes, and relations in a reality [10]. Strictly speaking from a computer scientist's point of view, an `ontology` is a formal, explicit specification of a shared conceptualization [11]. Computer scientists have used this definition to represent their concept of `ontology`, i.e. the representation of what exists in a specific application domain. The necessity was to mitigate the well-known problem of the Tower of Babel, in which different groups of data, designed by different designers

and experts, make use of different vocabularies to express similar or identical concepts. Vice versa, the same word may have multiple meanings, giving rise to a line of research in Natural Language Processing (NLP) known as "word disambiguation". In the early days, `ontology` designers sought to provide a complete classification of every concept available in the "universe" of knowledge. Not too far did this approach fail due to the overwhelming amount of objects to be classified and the variety of possible ways in which objects can be categorized according to the specific use, context, period and many other environmental factors. Last but not least, the objective of an `ontology` formalization is to provide a formalization that is "stable", as much as possible. For this reason, it should not be too tight to a specific context.

In the AI field, the aim is to create a model of the reality of interest. From a formal point of view, an `ontology` is a triple ($\mathcal{C}$, $\mathcal{R}$, $\mathcal{A}$) in which:

- $\mathcal{C}$ is a set of `concepts`.
- $\mathcal{R}$ is a set of `relationships` between concepts.
- $\mathcal{A}$ is a set of `axioms` on which the universe we want to describe is based [**?**].

$\mathcal{R}$ is therefore a subset of the Cartesian product $\mathcal{C}$ x $\mathcal{C}$. An `ontology` can also be devoid of axioms, in which case it is called non-axiomatized. An `ontology` can be basically seen as a set of concepts connected through relationships: hence, graphs are often used to describe them. A *graph* $\mathcal{G}$ is a pair ($\mathcal{V}$, $\mathcal{E}$) where $\mathcal{V}$ represents the set of `vertices`, in our case the concepts, while $\mathcal{E}$ is the set of `arcs` joining two vertices. In this case, $\mathcal{V}$ is the set of `entities` $\mathcal{C}$, and $\mathcal{E}$ is the set of `relationships` $\mathcal{R}$. Given this analogy, networks are the most common visual tool to represent `ontologies`. They represent an oriented graph in which each arc is labelled, and the label is, in a very intuitive way, the relationship between the two entities. We report a small example in Figure 2.1 showing some general concepts put into a hierarchy by the well-known "isA" relationship. This relationship creates a tree structure since a concept **Entity** always appears as root. Other relationships are **has_part** and **repairs**.

2.2 Knowledge Graphs

In the early days of AI, experts endeavoured to formalize all the knowledge related to a domain or use case. It is still the case now in some contexts but the support of

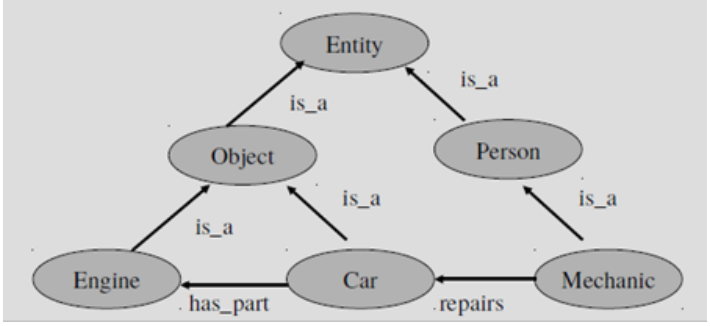


Figure 2.1: Ontology example

modern technologies contributed to automatizing part of the design or storage process. The term Knowledge Base (KB) refers to a collection of information expressed formally that is employed for several tasks, retrieval and reasoning as two of the most common examples. The discipline that investigated how to formalize information for reasoning tasks is KRR. It is the area of Artificial Intelligence concerned with how knowledge can be represented symbolically and manipulated in an automated way by reasoning programs [12]. KRR techniques took inspiration from human problem solving, and are to represent knowledge for intelligent systems to gain the ability to solve complex tasks. When knowledge is expressed in the form of a graph it is named Knowledge Graph (KG), which provides all the above-mentioned advantages of graph structures. A *KG* can be viewed as a KB in which we consider its graph structure [13]. The most used model for representing `KGs` is the `directed edge-labelled` graph (also known as a multi-relational graph [14–16]) is defined as a set of `nodes` such as "Santiago", "Arica", and a set of `directed labelled edges` between those nodes, such as "(Santa Lucía)-(city)->(Santiago)" [17].

More generally, a KG is a graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent potentially different relations between these entities [17]. `KGs` that represent structural relations between entities have become an increasingly popular research direction toward cognition and human-level intelligence [18]. They, as a form of structured human knowledge, have drawn great research attention from both academia and industry [17, 19–21]. The ever-increasing interest in this technology is due to its underlying abstract structure which effectively facilitates domain conceptualization and data management, and its usage as the main driver of several AI applications. In particular, the KG depicts an integrated collection of real-world entities that are connected by

semantically interrelated relations. In this respect, data are given formal semantics via data annotation and manipulation in a machine-readable format, thereby reducing ambiguity and deriving meaningful information that is specific to an application's domain. Therefore, the incorporation of KGs has extended the existing data models depicted by domain ontologies and established a new form of data analytics that can capture semantically interconnected, large-scale data sets [22].

It is necessary to define and differentiate KGs from other concepts to make valuable and accurate statements about the introduction and dissemination of these concepts. The second problem leads to the misleading assumption that the term Knowledge Graph is a synonym of Knowledge Base, which is itself often used as a synonym for ontology. An ontology is a formal, explicit specification of a shared conceptualization that is characterized by high semantic expressiveness required for increased complexity [23]. Ontological representations allow semantic modelling of knowledge and are, therefore, commonly used as Knowledge Bases in AI applications, for example, in the context of knowledge-based systems. The notion of KG stems from scientific advancements in diverse research areas such as the SW, databases, KRR, NLP, and machine learning, among others. The integration of ideas and techniques from such disparate disciplines presents a challenge to practitioners and researchers to know how current advances develop from, and are rooted in early techniques. The essential visualization elements involved in the notion of KGs can be traced to ancient history in the core idea of representing knowledge in a diagrammatic form. Examples include Aristotle and visual forms of reasoning, around 350 BC; Lull and his tree of knowledge; Linnaeus and taxonomies of the natural world; and in the 19th. Century, the works on formal and diagrammatic reasoning of scientists like J.J. Sylvester, C. Peirce and G. Frege.

In the beginning, studies were carried out in the fields of logic and knowledge engineering. The logic experts claimed that the whole complexity of the world could have been formalized in first-order logic and predicate logic-based languages. The goal was to infer correct conclusions starting from a KB made up of true facts and rules. This approach turned out to be limiting in many contexts so the rigidity was lightened with non-monotonicity [24] or fuzzy reasoning [25], capable of grasping the uncertainty and the fuzziness intrinsically present in the world. Differently from the first researchers, experts from the knowledge engineering field came up with new formalisms based on semantic networks, able to represent richer concepts, relationships and constraints among

entities and attributes. With the massive growth of the Internet, traditional methods based on artificially built KBs could not adapt to the amount of data available. For this reason, data-driven machine methods have become the current state-of-the-art in AI. Many practical implementations impose constraints on the links in KGs by defining a schema or ontology. KGs and similar structures usually provide a shared substrate of knowledge within an organization, allowing different products and applications to use similar vocabulary and to reuse definitions and descriptions that others create. Furthermore, they usually provide a compact formal representation that developers can use to infer new facts and build up knowledge [26]. Recent years have witnessed rapid growth in KG construction and application [21]. A large number of KGs, such as Freebase [27], DBpedia [28], YAGO [29], and NELL [30] have been created and successfully applied to many real-world applications, for solving tasks ranging from semantic parsing [30, 31] and named entity disambiguation [32, 33] to information extraction [34, 35] and question answering [36]. For some of these tasks, one of the most known KG is DBpedia. It is a community effort to extract structured information from Wikipedia and to make this information available on the Web. DBpedia allows you to ask sophisticated queries against datasets derived from Wikipedia and to link other datasets on the Web to Wikipedia data. We show a portion of the DBpedia as an example of KG in Figure 2.2.

KGs fit very well with problems concerning *reasoning*. Reasoning is one of the basic forms of simulated thinking and a process of deducing new judgements (conclusions) from one or several existing judgements (premises). Example of reasoning tasks over KGs aim to identify inconsistencies and/or incompleteness from existing data.

2.2.1 Graph Database

Relational databases have been providing storage support for many decades now with implementations like Oracle or MySQL. Way back people used databases just for storing tabular data like purchase reports and finance records. Relational databases were perfect as one could associate a transaction in the finance table with an item in the purchase table. In today's environment, these relational representations are inefficient in performing many operations [37]. The Web exhibits far more complicated networks of relationships than were expected when the Structured Query Language

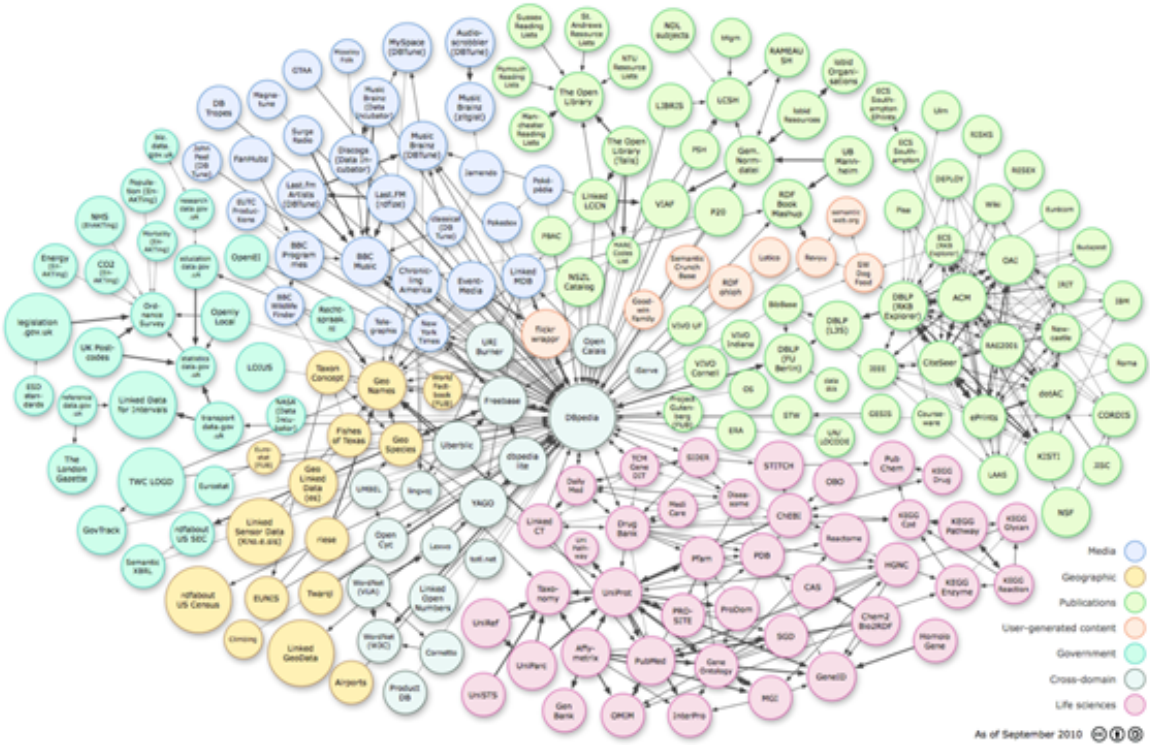


Figure 2.2: Portion of the DBpedia graph

(SQL) was designed. The network of hyperlinks connecting all the pages on the Web is highly complex and almost impossible to model efficiently in a relational database. Similar issues are involved in modelling social networks like Twitter and Facebook.

Implementing such problems in relational databases involves a large number of joins which is expensive to calculate. With the intense increase in usage of the internet leading to the need for storing large amounts of interconnected data, there was a clear desire for a data store tailored to the needs of graph data. `Graph databases` are optimized for these types of networks (social networking and website link structure), as a graph is a natural way of storing connections among users. Relational databases are not useful when the data model evolves over time, which means relational databases depend on rigid schema and make it difficult to add new relationships between objects. All these limitations of relational databases led to the development of NoSQL models.

A graph database is a database system where the relationships between objects or entities are equally as important as the objects themselves [38]. In a graph database, data are represented by nodes, edges and properties. `Nodes` are represented as objects and `edges` manifest the relationship between nodes. There are several implementations of graph databases. Both nodes and edges can have properties that depict their specific characteristics [39].

KGs use a graph-based data model to capture knowledge in application scenarios that involve integrating, managing and extracting value from diverse sources of data at a large scale. Employing a graph-based abstraction of knowledge has a number of benefits when compared with a relational model or NoSQL alternatives. Graphs provide a concise and intuitive abstraction for a variety of domains, where edges and paths capture different, potentially complex relations between the entities of a domain. Graphs allow maintainers to postpone the definition of a schema, allowing the data to evolve in a more flexible manner. Graph query languages support not only standard relational operators (joins, unions, projections, etc.) but also navigational operators for finding entities connected through arbitrary-length paths. Ontologies and rules can be used to define and reason about the semantics of the terms used in the graph. Scalable frameworks for graph analytics can be leveraged for computing centrality, clustering, summarisation, and so on, to gain insights about the domain being described. Promising techniques are now state-of-the-art for applying machine learning over graphs [17]. The term *data model* has been widely used in the information management community: it covers various meanings [40]. In the most general sense, a data(base) model is a collection of conceptual tools used to model representations of real-world entities and the relationships among them [41]. From a database point of view, the conceptual tools that make up a database model should at least address data structuring, description, maintenance, and a way to retrieve or query the data. According to these criteria, a database model consists of three components: a set of data structure types, a set of operators or inference rules, and a set of integrity rules [42]. Graphs have the advantage of being able to keep all the information about an entity in a single node and showing related information by arcs connected to it [43]. Associated with graphs are specific graph operations in the query language algebra, such as finding shortest paths, determining certain subgraphs, and so forth. Explicit graphs and graph operations allow users to express a query at a higher level of abstraction. It is not necessary to require full knowledge of the structure to express meaningful queries [44]. Finally, for purposes of browsing it may be convenient to forget the schema [45]. For implementation, graph databases may provide special graph storage structures, and efficient graph algorithms for realizing specific operations [46]. In the current era, graphs have been adopted in a plethora of domains like social, biological, and other networks. For instance, in biology, graphs are adopted to model genetic regulations

while in social networks they are used for modelling relationships between users. Graph databases implement the *Property Graph (PG) data model* [47]. In the PG model, the graph structure's elements can have some user-defined *attributes*. Attributes are key-value pairs available in the form of strings. Analysis of graph properties is deeply studied by the data mining community [48]. Graph database, like any other database model, follows CRUD (create, read, update, delete) methods [49]. There is a single data structure in a graph database – the graph – and there is no join operation so every vertex or edge is directly connected to other vertex. Graph databases shine when working in areas where information about data interconnectivity or topology is important. The following are a few examples of systems that would benefit greatly from the graph database technology. Recommender systems advise users on relevant products and information by predicting interest based on various types of information [50]. Graphs have seen extensive application in the field of bioinformatics, to relate a complex web of information that includes genes, proteins and enzymes. A prime example is the Bio4j project, which is a framework powered by Neo4j that performs protein-related querying and management. The project has aggregated most data available from other disparate systems into one golden source [51].

2.2.2 Labeled Property Graph

A standard extension of the PG model is the *Labeled Property Graph* model. In the LPG model vertices denote discrete objects and edges denote relationships between vertices. Both vertices and edges can have a number of key/value-pairs called "attributes". Keys and values are generic strings. Even though the performance of the LPG model has not been extensively studied [52], Neo4j, the most used graph database model, spread due to the competitive performance it provides to the community. Automatic sharding, built-in cache and replication are just some of its main benefits [53].

Graph databases provide better support for highly interconnected datasets than relational databases. Yet they are inherently schemaless, making them prone to data corruption, especially when users collect data from heterogeneous sources. Machine Learning algorithms that apply for generic KGs are also valid for this model. Wang et al. [54] proposed a new clustering algorithm based on properties of data.

Kalva et al. [55] provided the entire idea of how to retrieve the user's hidden

interests from a semantic network of intercorrelated contents on social networks. Formanowicz et al. [56] provided a short review of selected applications of graph databases in biology and chemistry.

2.2.3 Neo4j

Neo4j is an open-source graph database software developed entirely in Java. It is a fully transactional database, which is integrated into applications allowing standalone operation and stores all data in a folder. It is purpose-built to work with highly connected data, delivers lightning-fast performance and enables powerful, actionable insights.

Neo4j is a graph database that manages graphs and is optimized for graph structure instead of tables. Neo4j is the most popular graph database today. With Neo4j, you can map, store and traverse networks of highly connected data to reveal invisible contexts and hidden relationships. By intuitively analyzing data points and the connections between them, Neo4j powers intelligent, real-time applications that tackle today's toughest challenges. It combines native graph storage, a scalable architecture optimized for speed and ACID compliance to ensure the predictability of relationship-based queries, all without compromising data's integrity. Neo4j's index-free adjacency shortens read time, allowing you to run complex, multi-hop queries at lightning-fast speeds. Neo4j is accessible through Cypher, the world's most powerful and productive graph query language, but there is also API support for the most relevant programming languages.

Data in a database can be queried by a *Query Language (QL)*. A QL is a collection of operators or inference rules that can be applied to any valid instance of the data structures types of the model, with the objective of manipulating and querying data in those structures in any combinations desired [57]. Gremlin and Cypher are the two primary languages used to traverse Neo4j graphs. The first is a domain-specific programming language focusing on graph traversal and manipulation, while the latter is a declarative graph query language inspired by SQL that seeks to avoid the need to write traversals in code [51]. An example of graph database in Neo4j is shown in Figure 2.3

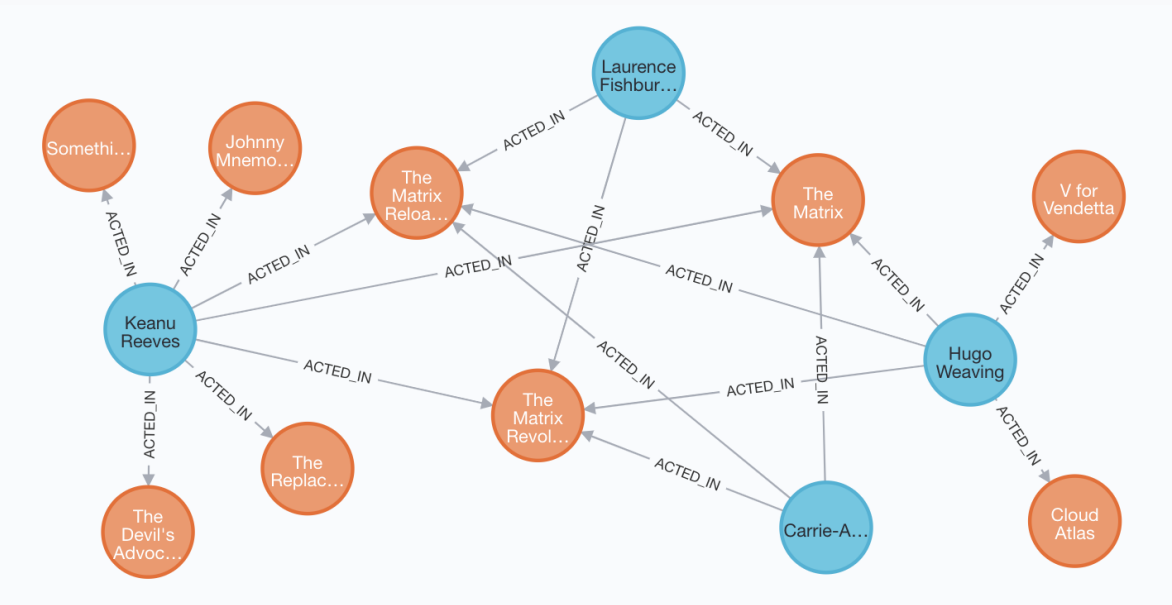


Figure 2.3: Example of a graph database

2.3 Semantic Web

The *SW* is usually envisioned as an enhancement of the current World Wide Web (WWW) with machine-understandable information (as opposed to most of the current Web, which is mostly targeted at human consumption), together with services - intelligent agents - utilizing this information; this perspective can be traced back to the 2001 Scientific American Article [58] which arguably marks the birth of the field. In the SW vision of the WWW, the content will not only be accessible to humans but will also be available in machine-interpretable form as ontological KBs. Ontological KBs enable formal querying and reasoning and, consequently, a main research focus has been the investigation of how deductive reasoning can be utilized in ontological representations to enable more advanced applications [59]. In the SW, metadata are generally in the form of ontologies, or at least a formal language with logic-based semantics that admits reasoning over the meaning of the data.

Over the years, it raised the question of what the difference between the Semantic Web and Knowledge Graph is. Smaller KGs, for example, enterprise Knowledge Graphs, can be differentiated from the Semantic Web because of their narrow domain. The goal of large search engines is to crawl and process all the information available on the web, which leads to increased interest in the widespread implementation of semantic technology. Considering the layers of the Semantic Web, a Knowledge Graph, by comparison,

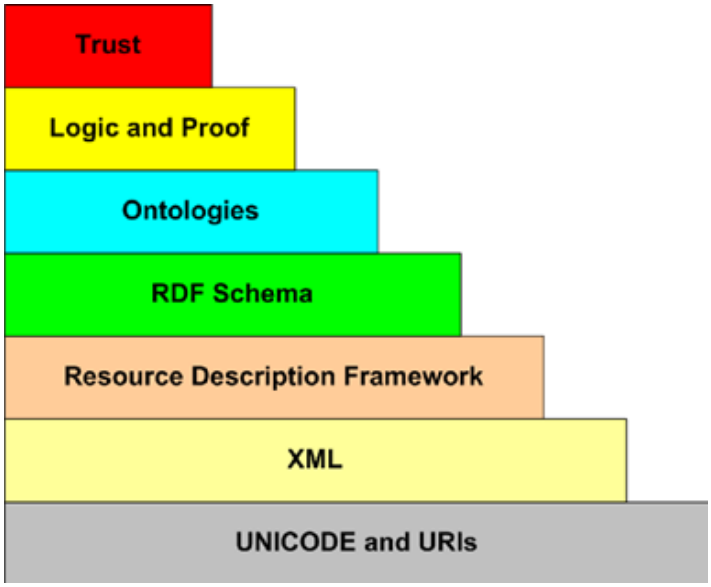


Figure 2.4: Semantic Web Architecture

employs either the same technology for each layer or a similar one that offers the same features. All in all, the `Semantic Web` could be interpreted as the most comprehensive `Knowledge Graph`, or, conversely, a `Knowledge Graph` that crawls the entire Web could be interpreted as an autonomous `Semantic Web`.

The `SW` is based on a set of technologies, tools and standards which form the basic building blocks of a system that could support the vision of a Web imbued with meaning. The `Semantic Web` has been developing a layered architecture, which is often represented using a diagram first proposed by Tim Berners-Lee, with many variations since. Figure 2.4 gives a typical representation of this diagram.

While necessarily a simplification which has to be used with some caution, it nevertheless gives a reasonable conceptualization of the various components of the `Semantic Web`. The layers are:

- `Unicode and URI`: Unicode, the standard for computer character representation, and `Uniform Resource Identifiers (URIs)`, the standard for identifying and locating resources (such as pages on the Web), provide a baseline for identifying resources.

- `XML`: eXtensible Markup Language (XML) and its related standards, such as Namespaces, and Schemas, form a common means for structuring data on the Web but without communicating the meaning of the data.

- `Resource Description Framework`: `RDF` is the first layer of the Semantic Web

proper. RDF is a simple metadata representation framework, using URIs to identify Web-based resources and a graph model for describing relationships between resources. RDF format is used to model relationships between two entities using a 3-tuple, called a triple. The 3-tuple consists of subject, predicate, and object. A predicate is a relationship between, or a fact about, the subject and the object. Subjects, predicates, and objects can be uniquely represented by URIs. All the elements in a triple can be URIs.

- RDF Schema: a simple type modelling language for describing classes of resources and properties between them in the basic RDF model.
- Ontologies: a richer language for providing more complex constraints on the types of resources and their properties.
- Logic and Proof: an (automatic) reasoning system provided on top of the ontology structure to make new inferences. Thus, using such a system, a software agent can make deductions as to whether a particular resource satisfies its requirements (and vice versa).
- Trust: the final layer of the stack addresses issues of trust that the Semantic Web can support [60].

Ontology Web Language (OWL) [61] is part of the growing stack of W3C recommendations related to the SW. OWL has a model-theoretic semantics that provides a formal meaning for OWL ontologies and instance data expressed in them. Several standards were proposed to formalize ontologies in the SW community. OWL turned out to be the main standard since it overcame the limitations of RDF Schema [62].

OWL embeds an inference engine, which adheres to the formal semantics in processing information encoded in OWL, with the objective of deriving new information from known information. An inference engine is needed for the processing of the knowledge encoded in the semantic web language OWL. An OWL inference engine has the following features:

- *Checking ontology consistency.* An OWL concept ontology (e.g., terms defined in the "Tbox") imposes a set of restrictions on the model graph. The OWL inference Engine should check the syntax and usage of the OWL terms and ensure that the OWL instances (e.g., assertions in the "Abox") meet all of the restrictions.

- *Computing entailments.* Entailment, including satisfiability and subsumption, are essential inference tasks for an OWL inference engine.

- *Processing queries.* OWL inference engines need powerful, yet easy-to-use, language to support queries, both from human users (e.g., for debugging) and software components (e.g., for software agents).

- *Reasoning with rules.* Rules can be used to control the inference capability, to describe business contracts, or to express complex constrictions and relations not directly supported by OWL. An OWL inference engine should provide a convenient interface to process rules that involve OWL classes, properties and instance data.

- *Handling XML data types.* XML data types can be used directly in OWL to represent primitive kinds of data types, such as integers, floating point numbers, strings and dates. New complex types can be defined using base types and other complex types. An OWL inference Engine must be able to test the satisfiability of conjunctions of such constructed data types [63].

Standards for representing KBs differ by their level of expressiveness. In figure 2.5, we find the possible categories of axioms that can be expressed.

OWL is SROIQ. The main source of intractability is the non-determinism. It requires a lot of guessing/backtracking. They are caused by:

- owl:unionOf.

- owl:complementOf and owl:intersectionOf.

- Maximum cardinality restrictions.

- owl:someValuesFrom and owl:allValuesFrom.

- Non-unary finite class expressions (owl:oneOf) or datatype expressions.

OWL is actually a family of three language variants (often called species) of increasing expressive power: OWL Lite, OWL Description Logics (DL), and OWL Full. OWL 1 Full is the most expressive variant of OWL 1. However, the free usage of vocabulary adds a source of undecidability; for example, OWL 1 Full does not enforce the well-known restrictions needed for decidability, such as using only simple roles in number restrictions.

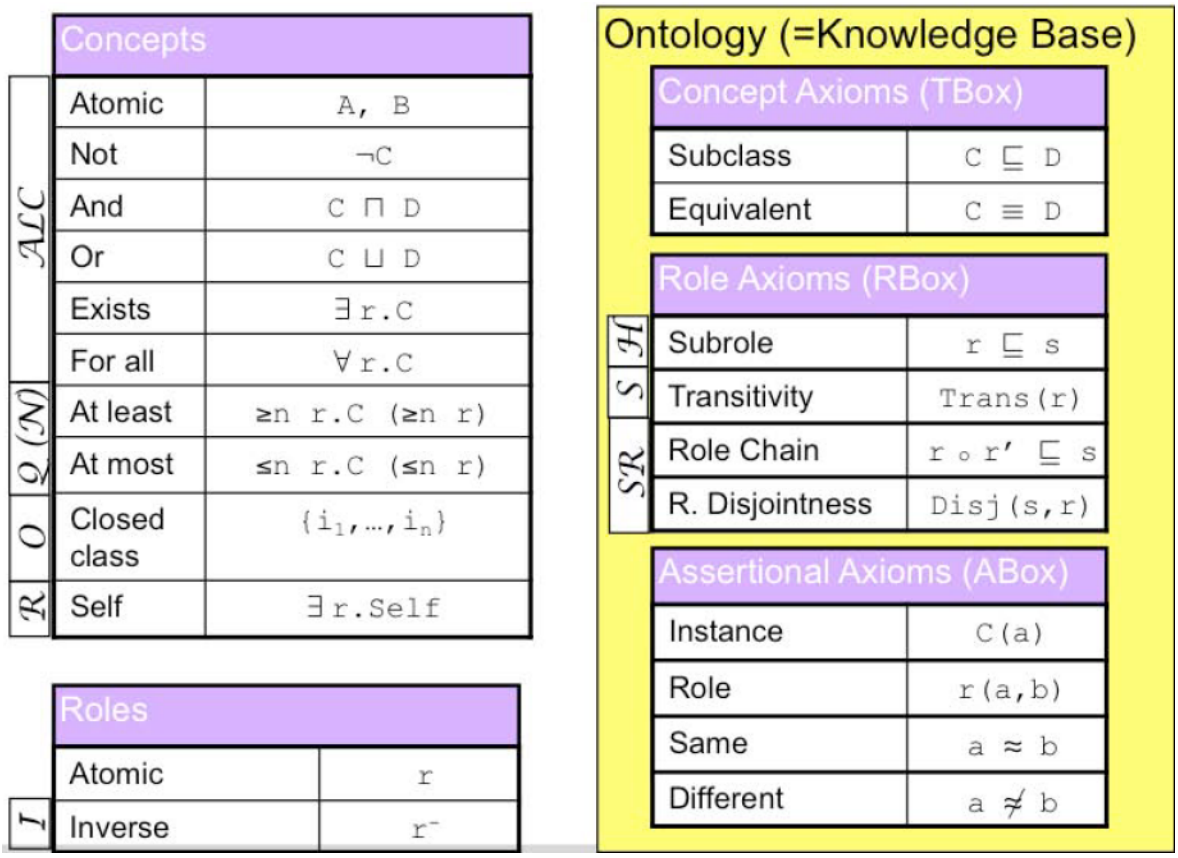


Figure 2.5: Levels of expressiveness

2.3.1 Resource Description Framework

RDF is a W3C recommendation designed to standardize the definition and use of metadata —descriptions of Web-based resources— but also to represent data. The basic building block in RDF is an object attribute-value triple, commonly written as $\mathcal{A}(\mathcal{O}, \mathcal{V})$. That is, an object $\mathcal{O}$ has an attribute $\mathcal{A}$ with value $\mathcal{V}$. Another way to think of this relationship is as a labelled edge $\mathcal{A}$ between two nodes, $\mathcal{O}$ and $\mathcal{V}$: $[\mathcal{O}]$–$\mathcal{A}$→$[\mathcal{V}]$. This notation is useful because RDF allows objects and values to be interchanged. Thus, any object can play the role of a value, which amounts to chaining two labelled edges in a graphic representation. RDF also allows a form of reification in which any RDF statement can be the object or value of a triple, which means graphs can be nested as well as chained. RDF Schema lets developers define a particular vocabulary for RDF data (such as authorOf) and specify the kinds of objects to which these attributes can be applied. In other words, the RDF Schema mechanism provides a basic type system for RDF models. This type of system uses some predefined terms, such as Class, subPropertyOf, and subClassOf, for application-specific schemas. RDF Schema expressions

are also valid RDF expressions [64]. The RDF data model uses URIs to unambiguously denote objects, and of properties to describe relationships between objects. The RDF data model distinguishes between resources, which are object identifiers represented by URIs, and literals, which are just strings. The subject and the predicate of a statement are always resources, while the object can be a resource or a literal. RDF graphs can be queried using the SparQL query language. SparQL [65] is a language that lets users query RDF graphs by specifying "templates" against which to compare graph components. Data which matches or "satisfies" a template is returned from the query.

2.3.2 Comparison and differences between LPG and RDF

The SW has been dominated by the RDF as W3C's standard model. Simultaneously, the concept of LPG, often using the native graph database Neo4j, is getting more attention lately. While different graph models proved to be useful in multiple use cases, there is a lack of empirical comparison of these graph models [66]. In Table 2.1 I summarized the main characteristics of the two graph models taken from [66].

Table 2.1: A comparison between the two graph models.

Graph model	Directed edges	Labels	Attributes	URI	Weights	Reasoning
LPG	✓	✓	✓		⊙	
RDF	✓	✓		✓	⊙	✓

As we can see from the table, most of the characteristics are in common and derive from the fact that they share the same data structure. When describing relationships, the concept of direction is fundamental to understanding which is the subject of the real relationship and what is the object (or value). Both of the models provide labels. This is not trivial since there exists approaches in which only PGs, not labelled, are used. Attributes are not present in RDF graphs. With LPG, we can put attributes to nodes and arcs, in the form of key/value. Values must be atomical and literal, otherwise, you should consider mapping it as a relationship between the node and another one having as a label the type of the attribute. Attributes in RDF are so reified. With reification, we mean to create a node having as the label the type of the value and create a relationship (probably named as the name of the attribute) between the two nodes. For this reason, OWL/RDF also provides types like Integer, String, Decimal and

so on, so that literals have their associated class to which they belong. There are many applications which need weights on arcs, or even on nodes. We can list just some of the most common algorithms used on the Web like Page Rank [67] or HITS [68]. Weights are not supported by any of the two models. Yet, it is still possible to represent weights by using attributes (in `LPGs`) or relationships (in `RDFs`). Handling weights in `RDF` becomes much heavier from a computational point of view because each triple should be reified and a relationship must be created to store the weight. Navigation will involve a much greater number of nodes. Finally, `LPG` graphs are just a different way to store data. They are thought in order to perform the CRUD operations on data. `RDF`, on the other hand, goes far beyond. It is developed to be exploited by a `SW` reasoner in order to infer new knowledge.

2.4 Reasoning

2.4.1 Ontological Reasoning

Reasoning in ontologies and Knowledge Bases is one of the reasons why a specification needs to be a formal one. By reasoning, we mean deriving facts that are not expressed in the ontology or the Knowledge Base explicitly. A few examples of tasks required from an ontological reasoner are as follows:

- *Satisfiability of a concept*: determine whether a description of the concept is not contradictory, i.e., whether an individual can exist that would be an instance of the concept.
- *Subsumption of concepts*: determine whether concept $\mathcal{C}$ subsumes concept $\mathcal{D}$, i.e., whether the description of $\mathcal{C}$ is more general than the description of $\mathcal{D}$.
- *Consistency of ABox with respect to TBox*: determine whether individuals in ABox do not violate descriptions and axioms described by TBox.
- *Check an individual*: check whether the individual is an instance of a concept.
- *Retrieval of individuals*: find all individuals that are instances of a concept.
- *Realization of an individual*: find all concepts to which the individual belongs, especially the most specific ones.

These tasks are not semantically very different. For example, satisfiability can be tested as a subsumption of a concept. It is unsatisfiable if no individual can exist that would be an instance of the concept. For all tasks, it is enough to be able to check deductive consequences or derive all deductive consequences of a theory. However, there may be specially optimized algorithms for different tasks in a reasoner.

All operations are decidable. Even when the theoretical complexity seems to be intractable, there are optimized reasoners available that are usable for practical real-world cases.

2.4.2 Logic Programming

`Logic Programming (LP)` is a simple, yet powerful formalism suitable for programming and for knowledge representation. It was introduced by R. Kowalski in the late 1960s. It grew out of an earlier work on automatic theorem proving based on the resolution method. The major difference is that LP can be used not only for proving but also for computing. It offers a new paradigm, which was originally realized in Prolog, a programming language introduced in the early seventies.

Prolog was originally designed as a programming language for NLP but it turned out to be used in every domain in which knowledge can be expressed in terms of facts and rules.

There are two natural interpretations of a logic program. The first one called a *declarative interpretation*, is concerned with the question *what* is being computed, whereas the second one, called a *procedural interpretation*, explains *how* the computation takes place. Informally, we can say that declarative interpretation is concerned with the *meaning*, whereas procedural interpretation is concerned with the *method*. The fact that, when designing a logic program, one can rely on its declarative interpretation, explains why LP supports *declarative programming* [69].

In LP programs, facts and rules are expressed using clauses. Giving a set of n predicates A_1, A_2, ..., A_n and a set of m negated predicates $\neg\ B_1$, $\neg\ B_2$, ..., $\neg\ B_m$, a clause is the disjunction of these literals (predicates or negated predicates): $A_1 \vee A_2 \vee ... \vee A_n \vee \neg\ B_1 \vee \neg\ B_2 \vee ... \vee \neg\ B_m$. Through the logical equivalence $\neg$ A $\vee$ B $\iff$ A $\rightarrow$ B, we infer: $B_1 \wedge B_2 \wedge ... \wedge B_m \rightarrow A_1 \vee A_2 \vee ... \vee A_n$ which means that a set of premises: B_1, B_2, ..., B_m lead to the conclusion $A_1 \vee A_2 \vee ... \vee A_n$. If n $<$ 2,

we define this clause as a Horn clause. An LP program is almost always expressed by Horn clauses. If n = 1, we have the rule in this form $B_1 \wedge B_2 \wedge ... \wedge B_m \rightarrow$ A. In this case, a set of premises B_1, B_2, ..., B_m lead to the conclusion of A.

2.4.3 Multistrategy Reasoning

Multistrategy reasoning concerns with extracting information from a Knowledge Base by following one (or more) strategies. The different strategies follow human reasoning methods that are feasible in any real-world context. We will briefly explain which inference mechanisms we consider part of multistrategy reasoning.

2.4.3.1 Abstraction

Abstraction is the type of reasoning aimed at removing unimportant details during the modelling of the solution. It solves different problems:

- the complexity of the problem is reduced.
- the problem becomes goal-dependent.
- problems considered impossible before are solvable.

In general terms, we can define the abstraction process as a mapping at the level of the perceived world. Formally, given a world W, let $\mathcal{R}_g = (\mathcal{P}_g(\mathcal{W}), \mathcal{S}_g, \mathcal{L}_g)$ and $\mathcal{R}_a = (\mathcal{P}_a(\mathcal{W}), \mathcal{S}_a, \mathcal{L}_a)$ be two reasoning contexts ground and abstract, an abstraction is a functional mapping $\mathcal{A}$: $\mathcal{P}_g(\mathcal{W}) \rightarrow \mathcal{P}_a(\mathcal{W})$ between a perception $\mathcal{P}_g(\mathcal{W})$ and a simpler perception $\mathcal{P}_a(\mathcal{W})$ of the same world $\mathcal{W}$. For "simpler" we mean that there is a function from the elements of the first perception to the element of the second one and this function is not injective.

2.4.3.2 Abduction

Abduction is the type of reasoning aimed at inferring causes from effects. Peirce defined it as the inference process of forming a hypothesis that explains given observed phenomena [70]. Its goal is to hypothesize unknown information starting from observations. It has the following characteristics:

- it handles missing information.

- different explanations are possible.
- many constraints must be satisfied.

More formally, given:

- a logical theory T representing the expert knowledge and
- a formula Q representing an observation on the problem domain,

An abductive inference is an explanation formula E such that:

- E is satisfiable with respect to T.
- T $\models$ E $\rightarrow$ Q

In general, we prefer E to be minimal (e.g. by restricting the number of predicates).

The abductive explanation of an observation is a formula which logically entails the observation and represents a cause for it.

We recall the difference between abduction and induction. The first one wants to infer causes from observation while the second wants to deduct consequences from observations.

- We provide an example:
  - If I say that my car will not start this morning, an *abductive solution* is the explanation that its battery is empty. An *inductive inference* can infer that if the battery is empty, then the car will not start.

2.4.3.3 Argumentation

Argumentation studies the processes and activities involved in the production and exchange of arguments. It aims at identifying, analyzing and evaluating arguments. It captures diverse kinds of reasoning and dialogue activities in a formal but still intuitive way and provides procedures for making and explaining decisions. In AI, it is used to give reasons to support claims that are open to doubt and/or defend these claims against attack. Argumentation takes a clue from everyday arguing which is something typical, involving different people who have their perspective which looks convincing.

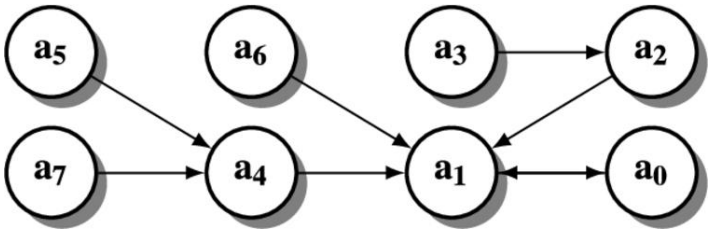


Figure 2.6: Argumentation example with the Dung model

In general, arguments are represented as nodes and relationships amongst arguments as arcs. It is easy to recognize a graph structure. Arcs are directed, and they indicate the source argument attacking the target one.

There are different frameworks for representing arguments. The basic one was proposed by Dung [71] and is shown in Figure 2.6 but it turned out to be too limited. Hence, the following ones emerged:

- *bipolar*: provides arcs also for supporting arguments, not only to attack.
- *weighted*: provides weights for attacks, to distinguish them.
- *trusted*: provides weight also for the arguments themselves.
- *mixed*: uses a combination of the previous ones, for example, bipolar and weighted.

Through this formalization, relevant aspects of the arguments can emerge, in particular some subsets of the overall set of arguments, like the conflict-free, the admissible, the semi-stable and so on.

Major applications of this kind of reasoning are debates, in which we can formalize arguments and attacks between them and, if using richer frameworks, possible weights, support relationships and others.

2.4.3.4 Analogy

Analogy is the process of transferring knowledge across domains. The main reasons for which it is used are the following:

- relevance in the study of learning, for moving from one domain to another.
- often used in problem-solving.
- relevance when studying new domains.

- in the past, it inspired great scientists in new discoveries.
- frequently used in communication.
- frequently used for explanations.

The analogy process is not straightforward. It requires the following phases:

- *retrieval*: finding the better base domain that can be useful to solve the task in the target domain.
- *mapping*: searching for a mapping between base and target domains.
- *evaluation*: providing some criteria to evaluate the candidate mapping.
- *pattern*: shifting the representation of both domains to their roles schema, converging to the same analogical pattern.
- *re-reprepresentation*: adapting one or more pieces of the representation to improve the match.

We provide a small example of an analogy process using the Knowledge Base, expressed in a graph in Figure 2.7.

In the image, the green arcs represent the relationships which do not require any mapping. Hence, a mapping is needed for the other ones and it is quite trivial:

- g $\rightarrow$ o.
- b $\rightarrow$ n.
- d $\rightarrow$ h.
- e $\rightarrow$ m.

Using the mapping and the information about the first domain, we can add missing information also to the second KB. In Figure 2.8, orange arcs have been mapped and purple ones have been added for analogy.

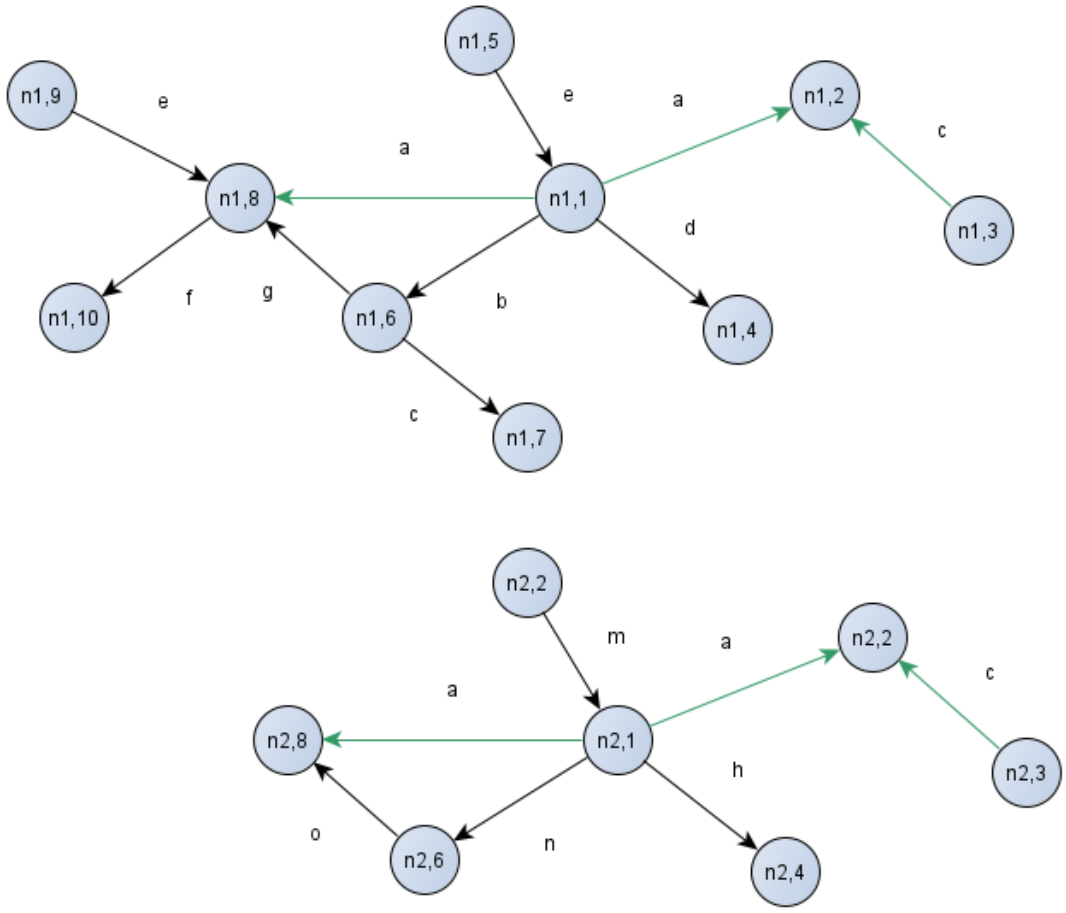


Figure 2.7: Example of two KB in analogy

2.4.4 Answer Set Programming

`Answer Set Programming (ASP)`, also known as Disjunctive Logic Programming (DLP) under stable model semantics, is a powerful framework for Knowledge Representation and Reasoning. Originating from the work of Gelfond, Lifschitz, and Minker in the 1980s, `ASP` has gained increasing interest within the scientific community. One key factor in its success is its highly expressive language: `ASP` programs can precisely express any property of finite structures over a function-free first-order structure that is decidable in nondeterministic polynomial time with an NP oracle, meaning `ASP` encompasses the complexity class $\Sigma_2^P = NP^{NP}$. As a result, `ASP` enables encoding programs that cannot be converted to SAT in polynomial time. Notably, `ASP` is entirely declarative (the sequence of literals and rules does not affect the outcome), and its encodings for a wide range of problems are concise, straightforward, and elegant. Unfortunately, the high expressiveness of `ASP` comes at the price of a high computational cost in the worst case, making the implementation a tough task.

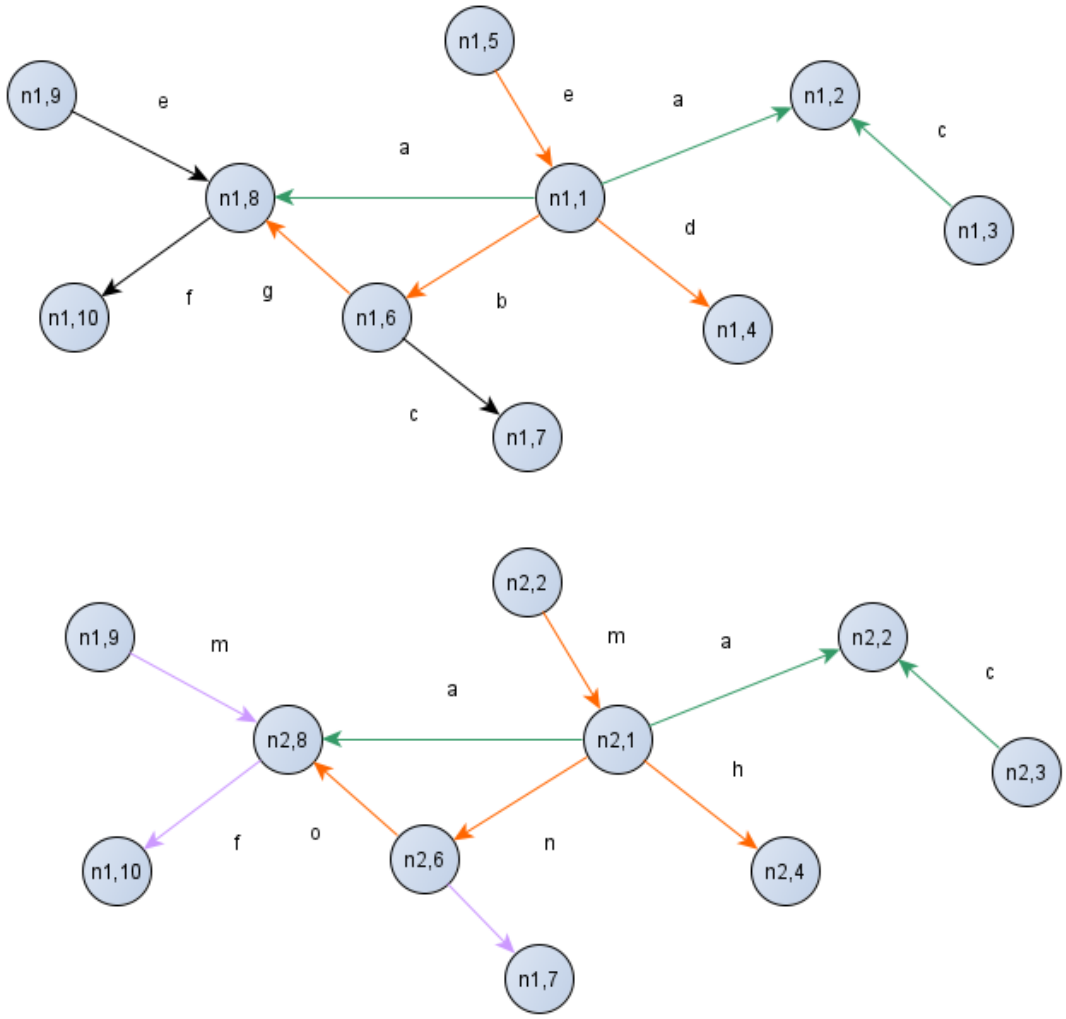


Figure 2.8: New KBs after analogy

2.4.4.1 Syntax

Following a convention dating back to Prolog, strings starting with uppercase letters denote logical variables, while strings starting with lowercase letters denote constants. A term is either a variable or a constant. An atom is an expression $p(t_1, ..., t_n)$, where p is a predicate of arity n and $t_1, ..., t_n$ are terms. A literal l is either an atom p (positive literal) or its negation $notp$ (negative literal). Two literals are said to be complementary if they are of the form p and $notp$ for some atom p. Given a literal l, $\neg l$ denotes its complementary literal. Accordingly, given a set L of literals, $\neg L$ denotes the set $\{\neg l \,|\, l \in A\}$. A set L of literals is consistent if its complementary literal is not contained in L for every literal $l \in L$. A disjunctive rule (rule, for short) r is a construct: $a_1 \vee ... \vee a_n \Leftarrow b_1, ..., b_k, \neg b_{k+1}, ..., \neg b_m$ where $a_1, ..., a_n, b_1, ..., b_m$ are literals and $n \geq 0$, $m \geq k \geq 0$. The disjunction $a_1 \vee ... \vee a_n$ is called the head of r, while

the conjunction $b_1, ..., b_k, \neg b_{k+1}, ..., \neg b_m$ is referred to as the body of r. A rule without head literals (i.e. n = 0) is usually referred to as an integrity constraint. A rule having precisely one head literal (i.e. n = 1) is called a normal rule. If the body is empty (i.e. k = m = 0), it is called a fact. If r is a rule of form (1), then $H(r) = \{a_1, ..., a_n\}$ is the set of literals in the head and $B(r) = B^+(r) \cup B^-(r)$ is the set of the body literals, where $B^+(r)$ (the positive body) is $\{b_1, ..., b_k\}$ and $B^-(r)$ (the negative body) is $\{b_{k+1}, ..., b_m\}$. An ASP program (also called Disjunctive Logic Program or DLP program) P is a finite set of rules. A not-free program P (i.e., such that $\forall r \in P : B^-(r) = \emptyset$) is called positive or Horn, and a v-free program P (i.e., such that $\forall r \in P : |H(r)| \leq 1$) is called normal logic program. In ASP, rules in programs are usually required to be safe. A rule r is safe if each variable in r also appears in at least one positive literal in the body of r. An ASP program is safe if each of its rules is safe. A term (an atom, a rule, a program, etc.) is called ground if no variable appears in it [72].

2.4.4.2 Semantics

From [73] the semantics of ASP programs can be defined. For every program P, its answer sets are defined by using its ground instantiation $grnd(P)$ in two steps: first, the answer sets of positive disjunctive programs are defined, and then, the answer sets of general programs are defined by a reduction to positive disjunctive programs and a stability condition. An interpretation I is a consistent set of ground literals $I \subseteq B_P$ w.r.t. a program P. A consistent interpretation $X \subseteq B_P$ is called closed under P (where P is a positive disjunctive datalog program), if, for every $r \in grnd(P)$, $H(r) \cap X \neq \emptyset$ whenever $B(r) \subseteq X$. An interpretation which is closed under P is also called a model of P. An interpretation $X \subseteq B_P$ is an answer set for a positive disjunctive program P, if it is minimal (under set inclusion) among all (consistent) interpretations that are closed under P.

Chapter 3

Related Works

This thesis touches on several aspects and, hence, has much literature in support. The related works are split according to the (related but distinct) areas in which the proposed work is placed.

3.1 Ontologies for domains of interest

I start here by describing the state-of-the-art ontological conceptualizations of several domains of interest, in which the proposed work can be applied.

3.1.1 Education

Today, open educational resources such as MIT's Open Courseware [74] provides free curated content [75], and location-aware search engines may offer course details in one's subject area. However, the realisation of truly personalised learning through technology is still to come. Many branches of AI are sufficiently mature enough to take significant steps towards this goal but have only been partially used for supporting educational purposes. Much research investigated LO recommendation [76], but usually following the same approach as commercial applications, and thus typically ignoring the learners' unique features and context, such as their background, history and preferences [77]. Some works tried to learn learners' profiles [78] generating behaviour's rules. However, to date, recommendations have been unable to guide the learners along the path necessary to attain their personally chosen goals as to goals specified by a teacher or school curriculum. Among the many proposals, only a few use ontologies to describe

LOs, user profiles, and context information together. Gasevic et al. [79] uses domain ontologies to annotate LO content, and content structure ontologies to enable direct access to LOs' components. Verbert et al. [80] investigated the interoperation of learning content defined according to different specifications. Koutsomitropoulos et al. [81] defined the basis for cross-repository semantics and proposed semantically enhanced characterization of LOs within a digital repository. Other initiatives, such as the one by Koutsomitropoulos et al. [82], rely on existing metadata schemas, e.g. Learning Object Metadata and Dublin Core. Some projects tried to expand the focus to other issues (e.g., the social aspects of education in IntelLEO[1]). Taking into account all these efforts, subjective aspects are almost always neglected. Concerning interaction between learners and the system, conversational agents have been proposed to answer questions asked by the learners to improve their learning path [83], and to provide suggestions on educational material [84].

3.1.2 Diachronic Analysys

Diachronic Analysis is the task of detecting semantic changes in words over time. One of the most common languages used for this task is Latin. Latin has one of the longest recorded histories of any human language, making it naturally suitable for quantitative studies [85, 86]. The first inscriptional records date from the sixth century BCE, and Latin continues to be used to the current day by the Catholic Church and some academic and legal institutions around the world. Written Latin diverged from the spoken vernaculars in the second half of the first millennium of the Christian era, but it remained in use as one of the principal channels of communication across most of Europe for the next thousand years. The humanists' conscious effort to reproduce Classical Latin led to a range of interesting developments, particularly affecting the neo-Latin lexicon to enable the expression of new concepts [87]. This extensive chronological span has raised the question of the extent to which Latin is seen as a dead or fossilised language [88, 89]. However, it remains an open question to what extent this fossilisation affected the semantics of words, as we know that the Latin lexicon, in this respect, has remained dynamic (over 4,500 words have acquired new meanings since the Renaissance). The extent to which post-classical Latin can be

[1] http://www.intelleo.eu/ontologies/activities/spec/

regarded as a "fixed" language [90–92] from the point of view of its ability to generate new meanings of words is still largely unknown beyond anecdotal evidence.

Empirical historical semantics research requires access to various sources, from dictionaries and lexicons to encyclopedic information and diachronic texts. While many have recognized the corpus-based nature of diachronic semantics, particularly for corpus languages like Latin [85, 93], quantitative corpus-based studies are yet to pervade historical semantics research. A critical barrier to this is that corpus and lexical resources for historical languages are siloed. While significant progress on linking lexical resources, tools, and corpora at the level of lemmas has been made [94], linking at the level of word senses is still missing. Given the significant progress made in the design of linked data models for language data [95], some studies [96] have already advocated for integrating corpus approaches with Linked Open Data technologies to study lexical semantic change. One crucial strategy for representing the results of research into language change as linked data is by modelling and publishing them as knowledge bases using a lexicon-based model, usually OntoLex-Lemon and its various extensions. This includes the Frequency Attestations and Corpus, FrAC, module, which proposes a new series of classes and properties for linking elements of a lexicon with corpora. Previous work in this area includes a proposal to modify the core organizing principles of wordnets to represent semantic shift phenomena [97], as well as work on the representation of etymologies as RDF graphs using OntoLex-Lemon [98] and the integration of temporal information into linguistically linked datasets via the so-called "four-dimensionalist" approach [99]. Integrating lexical resources and semantically- annotated corpus data at scale would allow us to gather corpus data on sense distribution information, essential for fully implementing the quantitative turn in historical semantics [100]. This integration, however, requires efficient handling of large datasets. In [101], a Linguistic Knowledge Graph was proposed, a model based on graph DataBase Management Systems (DBMSs). The Linguistic Knowledge Graph models relations between concepts and words, the information about word occurrences in corpora, and diachronic information on both concepts and words. The choice to focus on Latin was motivated by several factors. First, Latin has one of the longest recorded histories of any human language, making it naturally suitable for quantitative studies [85]; this, in turn, allows for corpus-driven analyses of semantic change processes over long periods. Second, this language has a particularly favourable position among historical languages

in terms of the availability of extensive corpus data in digital form (some of which have been linked to language resources and corpora at the level of word lemmas done in the context of the LiLa project) and computational language resources such as Latin WordNet [102] and digitized dictionaries. Focusing on the development of the Latin language, the contribution was the expansion of the range of Latin language resources that were included in the Linguistic Knowledge Graph for the study of lexical semantic change in Latin. The contribution included (i) the ingestion of Latin WordNet into the Linguistic Knowledge Graph; (ii) a new curated linking between existing resources for Latin, namely Latin WordNet [102, 103] and the SemEval Latin dataset [104, 105], a sense-annotated portion of the LatinISE diachronic corpus of Latin [106]; (iii) the integration of external contextual information (Wikidata) about the authors' occupation. There were also examples of using this integrated resource for lexical semantic change research.

3.1.3 Food and Health

The food sector has largely benefited from Artificial Intelligence. The applications are countless [107]. One of the first attempts to advertise meals based on the user needs was by Snae et al. [108]. It was based on Korean cuisine and represented a filtering system for diseases. Then, it proposed the fuzzification of the impact by assigning a certain degree of damage (not just a binary choice) and the provision of explanations by combining ingredients' features.

A more sophisticated recommender system by Toledo et al. [109] had personalization as it also considered user preferences. One of the main differences with the proposed approach was that it could not provide explanations for informing the user about the goodness of a choice. This is due to the use of interpretable graph models (graphs) and symbolic conceptualizations (ontologies).

Regarding the pure ontology designing phase, the attempt to unify food ontologies has been provided by Popovski et al. [110]. However, the unified conceptualization does not include peculiarities details for expressing the impact of the food and ingredients on allergies and/or diseases. In the work, the alignment took into consideration several existing ontologies such as FoodWiki [111], Open Food Facts [112] and others. Nonetheless, these resources cannot be compared with the richness of the (for example)

Italian websites.

In the context of food production tracking for sustainability and smart cities, Kamel et al. [113] integrated heterogeneous sources to combine information at different granularity and steps for a holistic production line of food.

In dealing with graphs, Qin et al. [114] constructed a Chinese cuisine knowledge graph with a query answering system retrieving resources gathered from the Web. In this case, the ontology is derived from data rather than manually constructed and refined for the task, also because the system was not supposed to be used for a specific use case. Again, the lack of correlation between food and illnesses, or the correlation with the ingredients' features is common. Without using conceptualization on a graph, as often happened with the LPG model, Bajaj et al. [115] introduced the use of Neo4j for the storage and recommendation of food. This represented a preliminary idea of our work in which the performance can be appreciated and justifies the ongoing research in this field with that representation.

Taking everything into account, many works aimed to recommend personalized food based on diseases/allergies, some of them are not explainable or limited in their interpretation since they did not make use of interpretable data structures or ontologies. The conceptualization of food barely took into account intolerances and the impact of each ingredient on them.

3.1.4 Digital Libraries

After the initial simple transposition to the digital format of library cards, with a pre-defined and fixed set of fields (or descriptive attributes), subsequent approaches to bibliographic records representation and management started leveraging the additional opportunities provided by digital technologies. Outstanding examples of this are the MARC format and the ISO 2709, currently a standard in the library practice. While fundamental, these are still deeply rooted in the record-based idea of library descriptions, in which the central and aggregating concept is the book or document, and every piece of information is considered as "belonging" to the book/document. A step forward toward relational descriptions was taken by the International Federation of Library Associations and Institutions (IFLA) with its Functional Requirements for Bibliographic

Records (FRBR)[2], a conceptual model of library information proposed in 1990. It provided an insight into non-obvious technical details of bibliographic description, at the same time widening the scope of descriptions. According to the foundational ideas from relational Database (DB) design, it identified a set of entities to be represented and described (e.g., places, persons, organizations) with their own "dignity", along with their characterizing attributes, and a set of relevant relationships among them that were implicit and hidden in traditional bibliographic records. So, the information enclosed in a bibliographic record is now distributed in a reticular information organization. The FRBR model has been refined and expanded in a later recommendation by IFLA, the Library Reference Model (LRM), proposed in 2017 [116, 117]. While important, these proposals still have significant limitations. First, and most importantly, they still focus on the syntactic level of descriptions, without providing formal definitions of the items (entities, relationships, attributes) and their behaviour in the library domain. This requires an upgrade from DBs to KGs, where ontologies provide such formal definitions, enabling interoperability among systems and platforms and advanced exploitation of that information by AI techniques. Second, the existing proposals still focus on information that is directly related to the documents to be described, and specifically to their formal metadata or properties, while the adoption of a "holistic" approach is preferred. A step by IFLA toward SW technologies was the Resource Description and Access (RDA) standard, whose main features are the extensibility of the descriptors, and the use of a formal representation that is closer to the ontological standards. Still, it could not be considered an ontology. In this direction: Hinze et al. [118] proposed one based on FRBR, ingesting the data from MARC sources; Alemu et al. [119] fostered a switch to open data, based on adapting to RDF the FRBR and RDA; Decourselle et al. [120, 121] focused on FRBRized entities, and specifically on the challenges related to metadata migration and semantic enrichment of bibliographic data. Many works in the literature highlighted the advantages of moving to a semantic-based approach to Digital Libraries. Semantic Digital Libraries [122] extensively used metadata to support information retrieval and classification tasks. They could use ontologies to organize bibliographic descriptions, represent and expose document contents, and share knowledge among users [123]. Moving from traditional descriptions, Babu et al.

[2]https://repository.ifla.org/bitstream/123456789/811/2/ifla-functional-requirements-for-bibliographic-records-frbr.pdf

[122] investigated how the Knowledge Organization Systems (KOS) can be integrated with the architectures using semantic technologies and data. Noting that the standard RiC-CM and CIDOC-CRM models used in library practice derive from the traditional approach used for the construction of catalogues, Bianchini et al. [124] proposed to connect the entities to the much larger, richer and more numerous ontologies of the SW, represented by the Linked Open Data (LOD) Cloud. However, it still focused on the elements of traditional schemas. Hallo et al. [125] depicted the situation of Linked Data technologies applied to libraries, outlining best practices, gaps and future trends based on the information found on authoritative library websites. Albeit often using RDF, most works did not use full-fledged ontologies, just controlled vocabularies and thesauri, such as OAI-ORE (Open Archives Object Reuse and Exchange), SKOS, Dublin Core, Newman's, MOAT [126], etc. Kruk et al. [123] analyzed some Semantic Digital Libraries projects, identifying three application areas of ontologies, and mentioned the MarcOnt Ontology [127] as a candidate for combining different metadata standards that could describe various concepts at different levels of granularity. Some works highlighted the need for an overall ontology that accommodated and "coordinated" all the different perspectives: Lacasta et al. [128] aimed at overcoming the limited semantics of thesauri and similar knowledge models proposing an automatic process to convert a knowledge model into a domain ontology through alignments with the lexical DB Wordnet and the upper-level ontology DOLCE [129]. To overcome the current limitations to the reuse of bibliographic data in the SW and of RDF languages, due to the lack of a common conceptual framework, Patricio et al. [130] proposed a reference model and a super-ontology to overcome the misalignments between existing bibliographic ontologies and the principles and techniques of LOD. Other works proposed ontologies that tackle specific aspects of digital library and bibliographic data management [131–133]. As for reasoning, Martin et al. [134] showed how a case-based reasoning system based on an ontology could enhance the effectiveness of information retrieval. Concerning the extent of descriptions, some attempts to widen it have been carried out mainly in the wider context of Cultural Heritage. Worth mentioning are CIDOC-CRM[3] and ArCo (Architecture of Knowledge)[4], an ontology and KG of Italian Cultural Heritage. It modelled many types of cultural properties (including technologi-

[3]https://www.cidoc-crm.org/
[4]http://wit.istc.cnr.it/arco/

cal heritage). Summing up, while there was a widespread feeling that ontologies were a need and an opportunity for modern and advanced Digital Libraries and bibliographic data management, the existing works often proposed tools, mostly aimed at improving Information Retrieval. The underlying data representations are almost always stuck to standard metadata (LAM or other), with those more oriented towards relational descriptions focusing on FRBR, sometimes proposing migration of data from the former to the latter, and only occasionally perceiving the need to expand them.

Overall, all extant approaches represented the text at some granularity but lost the full picture of its structure. The information they carry is mainly syntactic or semantic, while the layout and logical structure of the documents have not been so far completely considered. In most cases, they have been used to guide text preprocessing and splitting. Not only did the structure improve the effectiveness of document processing; but it allowed also the users to accomplish their tasks in a more precise and focused way, returning to them exactly the components they needed to satisfy their needs. Except for the lexical-conceptual mix, no attempt was present in the literature to describe documents from several complementary perspectives considered jointly and truly integrated, taking advantage of the full informative content of each. Instead, the focus was on the representation of (true) context. The "holistic" solution could be advantageous to fully express and exploit the documents' informative content, and to satisfy users' needs beyond what the current state-of-the-art may allow. The need to take into account the user's goals and domain knowledge was already recognized in [135]. The holistic approach aimed at taking even further those principles. Some works in the literature proposed to use different kinds of reasoning for this task. Croft et al. [135] supported both probabilistic retrieval strategies and browsing. Berger et al. [136] applied spreading activation on associative networks created from a relational database. Belew et al. [137] carried out probabilistic reasoning on a so-called connectionist representation based on weighted graphs, but using only features available in the database. Fox et al. [138, 139] proposed a Prolog system that indexes, retrieves, and recommends documents using logical rules to derive knowledge. In the context of Digital Libraries, one of the requirements that may be encountered is the need to distribute data. [140] tackled the task in the case of federated architectures by using a wrapper that acts as a query interface. Concerning the document structure, Popovici [141] targeted collections of heterogeneous multimedia XML documents, exploring methods for flexible

searching of structure, text, and sequential data embedded in such collections based on the structural part of the XML documents.

3.2 Ontologies and Databases

Integrating ontologies and databases is one of the most fascinating topics and has been founded on theoretical foundations since the early days of the discipline. The design of ontologies is somehow linked to the design of databases. Some design methods and techniques such as view integration that have been originally developed for large databases, where the "data models" and their semantics typically are limited to a particular application, could be relevant for this purpose. Meersman et al. [142] provided an overlook of the historical reasons and needs to move from one perspective to another. Historically, researchers have found similarities between the two perspectives and many solutions just mapped one prospect into the other by applying state-of-the-art and stable algorithms. The evolution of the DBMS models has led to the reconsideration of this idea while providing more advantages in terms of power and flexibility. Moving from a completely different model, like the graph, to knowledge bases from one point of view seems natural, but it also hides several complications which make this process demanding.

There have been approaches to query both databases and ontologies. Bulskow et al. [143] described an approach for query answering. The proposal is satisfactory as a process of wrapping results coming from different sources, but it does not combine the two aspects.

Calvanese et al. [144] described an approach based on DL for the representation of ontologies. The DL-Lite approach simulates ontology concepts in DL formalisms. In parallel, a relational database is designed for its representation, storage and management. This approach is very common as the relational logic model provides components that are mappable as components of ontologies. Relations represent classes, relationships represent their namesake concepts in ontologies. Attributes of relations represent class attributes. Descriptive logic has greater expressive power but, in certain simplified contexts, it could be developed in relational tables. The main shortcoming lies in the fact that the relational model is not able to represent inheritance. There are alternative methods for representing it, for example using a relationship specifically

for this concept. The mapping between ontologies and databases becomes more effective in Object-Relational DBMS (ORDBMS) or Object-Oriented DBMS (OODBMS) which took inspiration from the Object-Oriented (OO) programming paradigm. As such, inheritance is one of the key concepts and is therefore inherently available. The need to move from one level of representation to another may mean that in the future the two representations will become increasingly close in some DBMSs, to the point of even converging.

3.2.1 Graph Mining

The representation of the data schemas is preliminary for the subsequent phases in which the actual data are involved into. In the field of extracting information from a graph, we have a huge number of works. This is due to the fact that graph technologies have proven to be largely suitable for meeting market demands. Key strengths such as navigability, simplicity and flexibility have made them attractive in any domain. Mining graph databases is a very frequent task since the techniques exploit a lot of the navigability of the graphs. One of the most frequent task is to extract significant subgraphs [145]. Wang et al. [146] proposed an improvement of the gSpan algorithm [147] to find patterns in the data. The algorithm is called ADI-MINE, which is pattern growth. It consists of different steps. At the beginning, it builds an ADI (Adjacency Index) structure. It is performed by scanning the database of the nodes and arches only twice. In the first scan, the frequent edges are identified. According to the Apriori [148] property of frequent graph patterns, only those frequent edges can appear in frequent graph patterns and thus should be indexed in the ADI structure. In the second scan, graphs in the database are read and processed one by one. For each graph, the vertices are encoded according to the DFS-tree in the minimum DFS code, as described in [147]. Only the vertices involved in some frequent edges should be encoded. Each frequent edge is one of the smallest frequent graph patterns and thus should be output. Then, the frequent edges should be used as the "seeds" to grow larger frequent graph patterns, and the frequent adjacent edges of e should be used in the pattern growth.

In the case of finding recurring subgraphs from semi-structured data sets, there is the problem of the amount of patterns we can detect. Huan et al. [149] developed an algorithm for mining which avoids extracting patterns which are not part of other ones,

so that to reduce the results only to maximal patterns, reducing also the computational resources to be employed. The described framework is comprised of two tasks. The first is a graph partitioning method through which we group all frequent subgraphs into equivalence classes based on the spanning trees [150] they contain. The second important component is a set of pruning techniques [151] which aim to remove some partitions entirely or partially to find maximal frequent ones only. An overview of graph mining algorithms can be found in [152].

There have been attempts to perform reasoning from graph databases also in the field of LP. Many approaches [153] are based on relation databases that are easily mappable in Prolog-like languages. There are numerous applications in which this is valuable. Some of the first ones were implementations of recommender systems [154], or expert systems [155]. Starting from LP, different reasoning frameworks emerged. Especially in the current context of big data, surely there would be some incomplete or wrong data. We can handle these cases with the use of probabilistic inference [156, 157]. Graphs, seen in the SW perspective, opens the possibilities for ontological reasoning.

3.2.2 Reconciliation of RDF and LPG

In the literature, although in completely heterogeneous application domains, there have been several works that have tried to use the SW and graph databases simultaneously or that have created an intermediate layer to make one converge with the other.

Historically, this need has existed for some time. There are several works, even not too distant, that would have liked a method for connecting graph databases with RDF. In Zaki et al. [158], the huge amount of data available in traditional databases is mentioned, but mainly, the authors had to work with the integration of data already present in RDF format. The approach described, however, for integrating URIs coming from different sources is peculiar, but too tied with the biological field. Despite this, it can provide interesting insights into cases where similar techniques can be reused.

Interest expanded in the area of Natural Language Processing. Silva et al. [159] proposed a method to build a knowledge graph from a well-known lexical database: WordNet [160]. In particular, their aim was very similar to ours: that is, to build a new Knowledge Base that could be used for reasoning operations of various natures, in their case for Text Entailment Recognition [161]. Again, in the field of microbiology,

Penev et al. [162] presented OpenBiodiv: an infrastructure that combines semantic publishing workflows, text and data mining, common standards, ontology modelling and graph database technologies for managing biodiversity knowledge. It is presented as a Linked Open Dataset (LOD) [163] generated from the scientific literature. The data from both sources are converted to RDF and integrated into a graph database using the OpenBiodiv-O ontology and an RDF version of the Global Biodiversity Information Facility taxonomic backbone. This work lacks in the process of integrating (and disambiguating) knowledge coming from many sources but is a good example of an overall infrastructure dedicated for the integration. Purohit et al. [164] described a framework for the projection and validation of a portion of a graph in RDF using the graph database formalism. Specifically, the framework allows a SPARQL Protocol and RDF Query Language (SPARQL) query to be written, which translates the result into JavaScript Object Notation (JSON) and is parsed into the LPG model. At the end, the initial query is displayed in the new formalism. It is a useful tool for information verification and consistency checking with reference ontologies, but it is not usable for connecting the two tools.

There are also attempts [165, 166] to integrate SW techniques with relational databases using the fixed properties of the relations. Angles et al. [167] studied the RDF model from a database perspective. A lot of effort has been put into understanding the efficiency of languages for querying graphs. For instance, Sakr et al. [168] proposed a variant of SPARQL: G-SPARQL, which has its grammar and uses relational databases as a source base for finding data. Libking et al. [169] even proposed a new language. The goal was to introduce languages that work directly over triples and are closed, without using graph models. They also extend their language with recursion with good results when comparing with other languages. Thakkar et al. [170] proposed a new language: Gremlinator, which extends Gremlin [171] and can navigate graph databases from SPARQL queries.

Some approaches take one perspective and bring it to the other part. These approaches are more sophisticated since the translation from one technology into the other one is very complex and still an open issue.

The most frequent is the translation of RDF data into graph databases. Adding an intermediate layer, SPARQL queries are translated into Cypher and data can be retrieved as they are written directly in RDF. A proposal is described by De Virgilio et

al. [172] where the mapping translation follows an algorithm based on the structure of the nodes and an ontology, which acts as a reference for creating labels on each node. The following downsides emerged from the work:

- there is a loss of data at the end of the process, since, in a graph database, we do not have a link among other objects in the Web.
- for large KBs, there is a need of strong computational and memory power.
- after having translated RDF triples into the graph, we cannot perform any other operations but visualizing data.
- there is no way to capture potential inconsistencies after the data is inside the graph.
- transferring huge amount of RDF triples into LPG graphs requires a lot of computational time. This is one of the most underrated problems.

Tomaszuk et al. [173] proposed the system Yet Another RDF Serialization (YARS), which allows preparing RDF data to exchange on the property graph data stores.

Iordanov et al. [174] proposed a new graph database model. It is named HyperGraphDB and is based on generalized hypergraphs where hyperedges can contain other hyperedges. This generalization automatically reifies every entity expressed in the database, thus removing many of the usual difficulties in dealing with higher-order relationships. This approach does not use the tools of the SW at all. Hence, also the queries and operations possible with HyperGraphDB are not the same. There is still no graph model capable of covering all the advantages of the idea proposed by Tim Berners Lee.

Das et al. [175] considered the problem of supporting PGs as RDF in the Oracle Database. They introduce a PG to RDF transformation schema. They proposed three models:

- *Named graph based*: this proposal involves the use of quads (as opposed to triples) to create a unique named graph Internationalized Resource Identifier (IRI) for each edge. Then the label and the key/value properties of the edge are associated with the graph IRI.

- *Subproperty based*: id, label, and key/values for an edge can be modeled by creating a unique RDF property for each edge to represent the edge id, creating an RDF triple with that property as the predicate, associating the key/value pair with that property, and then making the property a subproperty of another property created based on the edge label.

- *Reification based*: in order to accommodate the id, label, and the key/value pairs for an edge, reification in RDF can create a new resource to represent every reified RDF statement.

This work lacks the reasoning part. The resulting graph can be queried in SPARQL.

Chiba et al. [176] integrated RDF triples in PGs in order to be exploited by some graph engines. There is an intermediate layer: G2GML, which follows an algorithm for bringing triples into the database. The drawbacks are the same as the similar approaches described above. There exists more sophisticated algorithms based on G2GML, like [177] which implements a better algorithm for the management of URIs and [178] which implements mapping rules for schemas and instances. Tomaszuk et al. [179] presented an ontology-based approach to transform (automatically) PGs into RDF graphs. The ontology, called PGO, defines a set of terms that allows describing the elements of a PG. The strength of this solution lies in the fact that it is not domain-dependent. What, from our point of view, can be improved is the possibility of performing translations from one model to another also taking ontologies into account in the translation process.

Schatzle et al. [180] proposed a mapping which is native to GraphX (a parallel processing system implemented on Apache Spark). The proposed graph model is an extension of the regular graph but lacks the concept of attributes. The mapping uses a special attribute label to store the node and edge identifiers, i.e. each triple t = (s, p, o) is represented using two vertices.

It should also be pointed out that one almost always ends up with data that are not complete. There are robust works that are well suited to these issues and have translation algorithms that are effective even in the most critical circumstances. An example of this is [181], in which some issues such as empty labels are handled.

There are also attempts to carry neither formalism to the other side, but to use an intermediate storage medium. This is the case of [182], which presents an effective

unified relational storage schema, that can seamlessly accommodate both RDF and PGs. Furthermore, it has been implemented the storage schema on an open-source graph database to verify its effectiveness.

It is evident that RDF and graph database systems are tightly connected as they are based on graph-oriented database models. On one hand, RDF database systems (or triplestores) are based on the RDF data model [183], their standard query language is SPARQL, and there are languages to describe structure, restrictions and semantics on RDF data (e.g., RDF Schema, OWL, SHACL [184], and ShEx [185]). On the other hand, most graph database systems are based on the PG data model, there is no standard query language and the notions of graph schema and integrity constraints are limited. Therefore, these two groups of systems (in particular the latter) are dissimilar in data model, schema, query language, meaning and content.

One of the main requirements to support syntactic interoperability is the existence of data formats (i.e., a syntax for encoding data stored in a database), over which transformation methods can be implemented. Turtle, TriG, RDF/XML, RDF/JSON and JSON-LD are data formats for encoding RDF data. In contrast, there is no data format to encode PGs. Given this restriction, some systems use graph data formats (like GraphML [186], DotML [187], GEXF [188], GraphSON [189]), but none of them is able to cover all the features presented by the PG data model. Hartig et al. [183] propose two transformations between RDF* (a syntactic extension of RDF which is based on reification) and PGs:

- mapping any RDF triple as an edge in the resulting PG. Each node has the "kind" attribute to describe the type of a node (e.g., IRI).
- distinguishing data and object properties. The former is transformed into node properties, and the latter into edges of a PG.

The shortcoming of this approach is that RDF* is not supported by the majority of RDF triplestores (except Blazegraph and the most recent addition, AnzoGraph) and requires conversion of existing RDF data beforehand. Semantic interoperability between databases means that both, source and target systems, can understand the meaning of the data to be exchanged. A common approach to support semantic interoperability is the definition of data and schema transformation methods. The schema transformation method takes as input the schema of the source database and generates

a schema for the target database. Similarly, the data transformation method allows the porting of data from the source database to the target database but takes care of the target schema. The transformation methods can be implemented by using data formats or data definition languages.

Nguyen et al. [190] proposed to combine the benefits into a single graph abstraction layer called Singleton Property Graph (SPG). The SPG layer sits on top of the RDF and simulates the Property Graph model. They describe the SPG model and its queries, which are Semantic Web-compliant, to be executed inside property graph databases. This middle layer can interpret SPARQL queries. In this work, however, they have not analyzed the problems related to control over the information that you can store in the graph database and the management of the corresponding semantic descriptions.

To the best of our knowledge, there is no method that support data and schema transformations between RDF and PGs keeping all their peculiarities. Based on our literature review about RDF and PGs interoperability, we identified several issues and challenges:

- there is no standard data format for encoding PGs. This is a crucial issue to support syntactic interoperability.
- the most RDF serializations are triple-centric, while the most PG serializations represent graphs as lists of nodes and edges.
- despite the serializations based on JSON or XML in both models, the syntaxes used are difficult to map.
- the support for multi-values is different in the models. A PG just support arrays, while RDF provides different types of lists.
- The RDF data model allows metadata about properties, i.e., edges between edges are allowed. Although this feature is not common in real data, a data mapping should be able to manage it. Note that a PG does not support multi-level metadata.
- RDF reification leads to an explosion in the size of the resulting graph. This can be avoided by implementing a "smart" transformation that is able to recognize a

set of triples describing a reification, and map them to a single node in the PG. [191]

Many other translation techniques from RDF to LPG are listed in [191]. Modoni et al. [192] provided a very detailed survey of the different techniques for storing RDF data and the peculiarities we can exploit from each of them. Hence, this thesis wants to propose an idea of integrating RDF tailored for graph storage and for the purpose of schema mapping, but future work in the field can also spread to other models or scopes.

3.3 Graph Schema Evaluation

Ontology design is an error-prone task, suffering from subjectivity and time dependency, following many strategies [193], some borrowed from traditional database design phases. Modern approaches emerged as soon as the technology and the amount of data changed. Literature provides plenty of works for automatic ontology extraction like *Map-Reduce* frameworks [194] tailored for Property Graphs [195], or dedicated frameworks for schema discovery and exploration [196]. Refinement requires choosing a valid (schema) updating technique [197]. In some cases, the need for ontology refinement is neither predictable nor desired (from the ontology designer's point of view). For instance, Palmirani et al. [198] developed an ML-based solution for an ontology to adapt to the new GDPR, making an evaluation purely based on concept mapping. Evaluating and refining ontologies has been explored from many perspectives. Historically, many attempts [199–201] exist to evaluate the properties and use of ontologies. However, these approaches are too tied to the respective domain knowledge, so no standard evaluation has been established. Ohta et al. [202] developed a framework in which the evaluation can be reproduced in different domains but lacks a global view since it starts from specific content and moves towards concepts following adjacent relationships. A more general evaluation was made by Jain et al. [203] who performed the evaluation from both an objective point of view, analyzing quantitative metrics, and a qualitative one by asking users to navigate through data based on the Emergency Situation Ontology and give feedback on the missing concepts.

A data-driven approach resembling our approach is based on hierarchical clustering

by Bonifati et al. [204]. Hierarchies fit with the need for ontologies as schemas. The difference to our approach lies in the metric used for clustering. Our simulation approach does not involve any numerical distance metrics. Although Angles et al. [205] recently provided a formalization for graph schemas, it does not support modal properties and is not standard. The new formalism involves the use of Cypher-like queries to define upper types on nodes and define properties and constraints on arcs. Such upper types are reflected by may-edges while we have limited further constraints with forbidden and mandatory properties. To date, Neo4j still provides little or no support for schema creation. Splitting graph data by using quotients has already been proposed for graph summarization tasks [206]. Our approach can be used for summarization in specific domains. Summarization is advantageous in plenty of tasks, and it is generally performed using bisimulation, a more strict condition of simulation. On the Web, it is often required to abstract away representation details in order to increase flexibility, reusability and interoperability. To this extent, new approaches emerge to build Web APIs based on graph structures. GraphQL [207] is a language for specifying constraints on nodes and arcs for property graphs. Basic constraints are covered, in order for GraphQL to be the equivalent of SHACL [208].

3.4 Graph Logic Programming

To date, solving graph problems in a logic-programming fashion is not common. However, some examples, like the reachability problem gained interest in many computer science areas, especially in some graph-based structures that were pioneers of AI like Petri Nets [209]. In general, this problem has been largely solved by database indexes [210]; however, finding alternatives and more general solutions is still valuable given that not all graphs belong to a database and, perhaps, new solutions may be integrated with indexes as well. From the general reachability problem, a class of subproblems emerged with more specific purposes like the k-hop reachability [211], and the shortest path problem [212]. Although historically there was a strict connection between logic and graph representation [213, 214], the problem of formalizing and solving the reachability problem with logic programs has been underrepresented. Heljanko et al. [215] formalized and solved the problem when designing deadlock detention, without introducing any specific optimization for reachability. A specific logic language [216]

has been provided for solving graph-based algorithms, but no optimized solutions were introduced in it. In the SW field, some restrictions can be overcome by introducing extensions with first-order logic for representing new graph rules [217]. With the introduction of uncertainty in logic programming, new applications become prone to be solved by logic inference. One of the most common regards recommendations with graphs [218]. The absence of a common schema governing graph-based instances gave rise to the introduction of Inductive Logic Programming [219] techniques to infer it [220]. Modern graph database strategies may perform a satisfiability check in a much more efficient way, but in [221] a logic for graph constraints definition was introduced. More interestingly, the types of constraints available in that work are quite similar to those currently available in Neo4j.

3.5 Existing Tools

The state-of-the-art, GraphBRAIN, which we have mentioned above but will describe better later, uses both frameworks for different purposes, mainly making use of the OWL API for reasoning operations. The two frameworks can be used independently. In this section, I briefly recall some available online systems that support the representation of semantic KGs following the SW standards.

3.5.1 W3C RDF Validator

The first, simplest, tool is software created by W3C for validating an RDF document. It is much more than a validator as it also builds triples and a Knowledge Graph. It is a parser that can read an RDF document and construct, after syntactical verification, the triples related to the knowledge graph. By setting some parameters it is also possible to choose if you want to show the relative graph and its format.

3.5.2 Apache Jena

Apache Jena is a free and open-source Java framework for building Semantic Web and Linked Data Applications. It can interact with the core API to create and read RDF graphs and serialize triples using popular formats such as RDF/XML or Turtle. It gives the possibility of querying your RDF data using ARQ, a SPARQL 1.1 compliant

engine and reason over your data to expand and check the content of your triple store.

3.5.3 OWL API

The OWL API is a Java API and reference implementation for creating, manipulating and serializing OWL ontologies.

The OWL API is open source and is available under either the LGPL or Apache Licenses.

Unluckily, at the state of the art, the explanation mechanism is not well documented and enriched. For this reason, explanations are not always available (e.g., in some inconsistency problems).

3.5.4 Neo4j Plugins

Neo4j also has libraries for the automatic import of files in different formats, including OWL/RDF. In support of this need the library "neosemantics" allows importing and manipulating functions.

Chapter 4

GraphBRAIN (State-of-the-art)

The work that I used as a starting work to achieve the goals is GraphBRAIN [222], an existing technology which manages ontologies and arranges data in a graph database. This technology is the starting point for knowledge storage and representation. I devote this chapter to the description of GraphBRAIN before starting the proposed work.

GraphBRAIN [222] is an ontology management system to create, extend, import, and modify ontologies from different sources. It is possible to choose one of the possible available domains (food, general, lam, retro-computing and tourism) and visualize entities (or classes), relations and a graph of nodes. The goal is to encapsulate as much knowledge about the world as possible, and this is accomplished by the users who can easily register and enter new information. There is also a Hall of Fame divided by domains where you can see who the major contributors to the service are; it is also designed to become a tool to store the reliability of each cooperating user. You can learn about the different entities, see their values and insert new ones by specifying some mandatory attributes (e.g., the *name*). You can also identify the relationships between each entity. The structure of the relations is simple since they have a name, subject and object. The subject represents the orientation of the relationship or the class label that starts the arc in the case of networks. Every relation has an inverse one in which the subject and object are reversed. Entities and relationships can also be described by attributes that add additional information, as in the LPG setting. GraphBRAIN is based on a graph database that is much more efficient than a relational one, especially in the case of a large amount of data. There is also a section where you can identify in a graphical way some nodes and relations expressed through labelled nodes and arcs.

The editing and management of GraphBRAIN Schema (GBS) is available as a web applications based on the Java Server Faces technology and the PrimeFaces library. JavaScript was used to handle interactive graph browsing. Clearly, Neo4j is used to store the knowledge graph, while Postgres is used to store user and usage data (roles, access rights, changelog, etc.). A demo of the tools can be found at `http://193.204.187.73:8088/GraphBRAIN-detached/` in the form of a general-purpose system for the collaborative development, management and (personalized) fruition of a KB. After logging into the system, the user may choose a domain and all subsequent interaction is driven by the corresponding GBS schema.

4.1 Ontology Management

When defining their ontologies, users can see the classes and relationships of existing ontologies and may reuse them, possibly extending or refining them, or defining super- or sub-classes thereof. In particular, GraphBRAIN provides a top-level ontology, defining very general and highly reusable concepts (e.g., **Person**, **Place**) and relationships (e.g., **Person.wasIn.Place**). It acts as a hub and plays a crucial role in interconnecting the domain-specific ontologies, ensuring that a single, overall connected, Knowledge Graph underlies all the available domains. Particularly interesting is class **Category**, aimed at storing items from different taxonomies as individuals in the Knowledge Graph. This allows handling them within the graph. It includes the WordNet lexical ontology [223] (also using class **Word** for its lexical part) and the standard part of the Dewey Decimal Classification (DDC) system [224]. So, **Category** and **Word** nodes may be linked by arcs to individuals of other classes (e.g., documents, persons, places) and used as tags to express information about them. Specifically, words may be used for lexically tagging other items (e.g., this paper, as a **Document** node, might be linked to "graph", "ontology", etc.), while categories may be used to semantically tag them (e.g., this paper might be linked to "Computer Science", "Artificial Intelligence", etc.) without formalizing thousands of classes in the ontologies. **Category** and **Word** nodes are also related to each other, thus enabling a form of reasoning as graph traversal and allowing to find non-trivial paths between instances of other classes. The ontologies are formalized in a customized XML format, purposely designed to be used as a schema for the graph database. However, techniques for exporting (and mapping) GBS onto

GBS and LPG Mapping	
GBS Element	LPG Component
entity instance	node
relationship instance	arc
entity name	label
relationship	name type
domain name	label
entity attribute	node property
relationship attribute	arc property

Table 4.1: Mapping from GBS to LPG.

Semantic Web formats are available. Currently, serialization to OWL format is provided, so that they can be published and exploited to ensure semantic access to the knowledge base and make it interoperable with other resources.

4.2 Mapping Database and Ontology

Part of the main motivation for defining schemas is to endow LPG-based graph databases with a schema that ensures clear semantics to the information pieces they contain and provides directions for their management and interpretation. According to this perspective, the database users will be required to work according to pre-specified data schemas expressed in the form of ontologies. Operationally, the database will be wrapped into a layer, e.g., in the form of an API, that takes as input a schema specifying the desired domain ontology and controls all interactions, allowing the external applications to manipulate and consult only information items that are compliant with the ontology. In this approach, they also provide an additional opportunity. Specifically, they allow a single graph database to underlie several domains (schemas), provided that their elements (entities and relationships) are compatible. By compatible I mean that the same classes are equally named in the different schemas, or the same attributes have the same name and datatype in different schemas. Non-shared elements can be freely defined. Therefore, using any of such schemas on the database would provide a partial view of its contents, perhaps representing a different perspective or aimed at limiting access to the database contents for some users or applications. Table 4.1 summarizes the interpretation of GBS elements with respect to the LPG ones.

4.2.1 Entities and Relationships

Leveraging the possibility of using many labels for nodes, each node is labelled with the top-level entity it belongs to and with all the domains for which it is relevant (e.g., 'Herbert Simon' would be labelled with "Person" for the entity and with and "retrocomputing" for the domains). When the same database underlies several domains, this allows selecting only the instances involved in a domain of interest. On the other hand, since each arc may take at most one type, we use it for specifying the relationship it expresses. The domains for which a relationship instance is relevant may be inferred from the domain labels of the nodes it connects by considering all the domain labels that are present in both its subject and its object.

4.2.2 Attributes

Concerning attributes, we propose to reserve an attribute name ('value') to store which is the specific sub-entity (resp., sub-relationship) the entity (resp., relation) instance belongs to. Given the top entity (resp., relationship) specified in the labels (resp., types) and the specific sub-entity specified in the "value" property, the path of specializations between these two may be easily recovered bottom-up starting from the latter and climbing the specialization hierarchy in the ontology up to the former (since nodes admit many labels, one might specify all the sub-entities in such a specialization path as labels; for the sake of uniformity with arcs).

4.2.3 Attribute Types and Values

Attribute values of types integer, real, boolean, string and text are stored as literal values for the corresponding database types, e.g., Neo4j provides the following types matching GBS types: Integer and Float (both subtypes of an abstract type Number), Boolean, and String. For types "select" and "tree" the string corresponding to the selected value in the list or tree is stored. An attribute of type "entity" will be represented in the graph as a relationship between the current instance and an instance of the target entity, connecting the nodes corresponding to these two instances and having the attribute name as the type. In the proposed naming policy attribute names start with a lowercase letter, just like relationship names [225].

4.3 Main Functionalities

GraphBRAIN provides several analysis, mining and information extraction functionalities, including tools to:

- assess the relevance of nodes and arcs in the graph and extract the most relevant ones (currently: Closeness centrality, Betweenness centrality, PageRank, Harmonic centrality, Katz centrality).
- extract a portion of the graph that is relevant to some specified starting nodes and/or arcs (currently: Spreading Activation and a proprietary variation thereof).
- extract frequent patterns and associated sub-graphs (currently using the Gspan algorithm and a variation thereof that is bound to include specific nodes in the graph).
- predict possible links between nodes (currently: Resource Allocation, Common Neighbors, Adamic Adar, and a proprietary approach).
- retrieve complex patterns of nodes and relationships (currently based on the application of Prolog deduction, using user-defined or automatically learned rules, on selected portions of the knowledge graph, suitably translated into Prolog facts).
- translate selected portions of the graph into natural language (using a Prolog-based engine to organize the selected information into coherent and fluent sentences).

Some of the underlying algorithms are reused from the literature; others have been purposely extended to improve their ability to return personalized outcomes that may better satisfy the user's information needs, which is another novelty introduced by GraphBRAIN [222].

4.4 GraphBRAIN Schema

GraphBRAIN provides the possibility of creating and/or updating ontologies online. This is fundamental for the subsequent operations and methodologies we aim to es-

tablish. We want to allow building complex relationships to provide a high level of abstraction. The current schema is an evolution of the one described in [225].

Schemas in GraphBRAIN must follow a format named the GraphBRAIN Schema format. The following concepts must be represented to exploit all the potentialities of LPG graphs: a hierarchy of entities, relationships and attributes on both entities and relationships. I extended [225] to represent also sub-relationships and user-defined types. Schemas not only abstract what LPG allow to, but also expand their capabilities introducing the possibility of merging schemas, defining user-defined types and so on.

Entities and sub-entities represent the basic elements of ontological schemas. Hierarchies of entities and relationships follow the inheritance principles of the OO paradigm. Relationships have to specify domain and range, to which we will refer as "subject" and "object" [64]. The peculiarity is that users can specify all the possible subject-object pairs for which the relationship holds. Sub-relationships inherit all subject-object pairs from the parent, intuitively.

Both entities and relationships may have attributes. Attributes (properties) can take values from the basic (primitive) data types (integer, string, date, boolean, text, and float) or instances of a user-defined entity.

User-defined types can be specified. When needed, they can be referenced in the attributes by their name. As an example, one may need to use the attribute "gender" many times (e.g. on people, and animals).

Each GBS schema is intended to describe one domain. However, sometimes wider domains involve ontological elements that are already described in more "basic" schemas (e.g., the schemas for Cultural Heritage, Food and Transportation might be exploited in the ontology aimed at supporting a touristic application) and it might be useful to reuse such schemas, both for standardization of the definitions and for building on existing knowledge. We will describe the possibility of importing ontologies in Section 4.4.3.

4.4.1 Predefined Entities

Before mentioning how to name and specify ontological concepts, I mention some predefined concepts which need to exist regardless of the specific domain or design. The specific class is called *TimeSpecification*. It acts as an abstract class because it provides

subclasses like *Year*, *Month* and *Day*. To specify a specific date, we must specify a *Day*, that belongs to a *Month* of a specific *Year*. We may not have information about the specific date or we care about a specific period of time limited by months of the same year. So we have the possibility to use only *Month* and *Year*. This *TimeSpecification* is fundamental when modelling periods, temporal arcs, historical trends or facts.

4.4.2 Formalism

Here is described how to formally express schemas. GBS schemas are expressed in XML because it is easy to manage and modify, not overly complex, and flexible.

A schema consists of an XML file whose tags provide all the kinds of components in GBS ontology. In the following, when specifying the GBS file structure, we will adopt the usual notation of square brackets [...] to denote optional elements, curly brackets {...} to denote repeated elements and pipes in parentheses (... | ...) to denote choices. Furthermore, we write XML tag names in boldface, XML tag attribute names in italics and entity or relationship names in small caps. Text in plain typeface reports comments useful for understanding the various elements and their behaviour.

The main structure of a GBS file with the tags and their nesting is reported in Table 4.2, where the universal entity ENTITY and the universal relationship RELATIONSHIP, acting resp. as the roots of the entity and relationship hierarchies, are implicitly assumed (using the SW terminology, we would refer to classes with "entities" and to object properties with "relationships"). Therefore, entities and relationships are to be specified only starting from the first level of specialization, which we will call *top-level*. For any entity or relationship in the hierarchy, its direct specializations are considered as disjoint and are not to be intended as a partition: instances that do not fit any of the specializations of a parent (sub-)entity or (sub-)relationship may be directly associated with the parent. Therefore, also the top and intermediate levels of each hierarchy admit instances in the KB, apart from the roots. The only exception is "abstract" entities (or relationships) that cannot have instances. This design choice prevents multiple inheritances (associating an instance to many classes belonging to different branches in the hierarchy). We partially recover this at the level of instances: when two instances of different (sub-)entities (or relationships) represent the same object, they are linked by an ALIASOF relationship. The single reference

object represented by all these instances ideally takes the union of their attributes.

Before specifying entities and relationships, two optional sections can be filled. The former allows importing another ontology into the current one. This is fundamental in order to create modular pieces of knowledge which can be combined. Imports may also be used to add further information about previous concepts. If an ontology specifies an already-imported concept, the differences will be tried to be merged, if possible, according to the strategy defined in Section 4.4.3. The import is transitive, that is, an imported ontology may import other ones.

In the subsequent sections of GBS, the user can define new types to be used in the entity or relationship attributes. In this way, the level of abstraction increases and reuse is enforced.

domain // tag enclosing the overall ontology

[**imports**]
[**user-defined types**] // see **(X)**
entities // tag enclosing the classes
{**entity**} // see **(*)**
relationships // tag enclosing the relationships
{**relationship**} // see **(*)**

Table 4.2: Main structure of GBS files.

The structure for describing user-defined types is shown in Table 4.3. The type can be "tree" or "select". The first represents a hierarchy of values that a specific attribute can be associated with. The latter represents an attribute whose values are predefined, and specified in the subsequent "value" section.

(X) (**type ... datatype="tree"** | **datatype="select"**) tag

values
{**value**} // see **(**)** (recursive)

Table 4.3: Structure for describing user-defined types in GBS files.

The structure for describing entities and relationships is specified in Table 4.4. Tag **references** is used only in relationships to specify their possible subject-object pairs. **taxonomy** is optional (used only if the entity or relationship has sub-entities or sub-

relationships) and allows us to conveniently represent the specialization hierarchy (i.e. the entities hierarchy, or the relationships one in **relationships** section). **attributes** is mandatory for non-abstract entities (their instances must be described by some attribute) and optional for relationships (a relationship may carry information in its very linking two instances). **value** is a recursive tag, allowing to define hierarchies of sub-entities or sub-relationships. In addition to its own attributes each specialization inherits all the attributes of the entities (resp., relationships) on the hierarchy path from its specific **attributes** section up to the corresponding top-level entity (resp., relationship). Clearly, subjects and objects of sub-relationships must be the same as the subjects or objects of their parents, or subclasses thereof.

(*) (**entity** | **relationship** | **value**) tag
[**references**]
{**reference**}
[**taxonomy**]
{**value**} // see **(*)** (recursive)
[**attributes**] specifying the data properties
{**attribute**}

Table 4.4: Structure for describing entity and relationship hierarchies in GBS files.

Some tags have XML attributes that specify the details or properties of the item they represent in the schema:

- **domain** tag:

  name the unique identifier for the domain being described

  author the author of the schema

  version the version of the schema

- **entity** tag:

  name the unique identifier for the entity

- **relationship** tag:

  name the unique identifier for the relationship

inverse the unique identifier for the inverse relationship of *name*

- **reference** tag:

  subject the identifier of the entity that is the domain of the (sub-)relationship

  object the identifier of the entity that is the range of the (sub-)relationship

- **value** tag:

  name the unique identifier for the specialization (sub-entity or sub-relationship)

- **attribute** tag:

  name an identifier for the attribute

  mandatory = (**true** | **false**)
  whether the attribute must take a value in each instance

  distinguishing = (**true** | **false**)
  whether the attribute may concur in distinguish instances having the same values for mandatory attributes

  display = (**true** | **false**)
  whether the attribute represents interesting additional information with respect to mandatory and distinguishing attributes, to be possibly displayed

  datatype = (**integer** | **real** | **boolean** | **string** | **text** | **select** | **tree** | **date** | **entity**)

  [***length***] the maximum allowed number of characters (used only when datatype = string)

  [***target***] an entity name (used only when datatype = entity)

The union of mandatory and distinguishing attributes of an entity or relationship can be used to specify a key for uniquely identifying its instances. The union of mandatory, distinguishing and display attributes of an entity or relationship can be used to build and display a summary reporting the most relevant information about the instances.

Regarding datatypes, attributes of type *integer*, *real*, *boolean*, *string*, *text* take an atomic value of the corresponding type, where *text* is intended for the free text of

any length, differently from *string* which has a limited maximum length that can be specified in the 'length' attribute. Attributes of type *date* take values in one of the following forms:

- Year;
- Year/month;
- Year/month/day.

In Section 4.4.1 we already mentioned the *TimeSpecification* entity and it represents our design choice for memorising dates. The year is any integer, month $\in \{01, \ldots, 12\}$ and day $\in \{01, \ldots, 31\}$. Values of type *date* are instances of entities of the above-mentioned predefined entities *Year*, *Month* or *Day*. This means that this attribute is modelled as a relationship in the graph, similar to what happens to attributes of type "entity" (see below) that have a user-defined type. Attributes of type *select* denote a choice in an enumeration of values, described using the substructure reported in Table 4.5; attributes of type *tree* denote a choice in a tree of values, described using the recursive substructure shown in Table 4.6. Attributes of type *entity* denote 1:1 (i.e. "functional") relationships between an instance of the current entity and an instance of another entity (specified in the 'target' attribute of the tag), e.g., the birthplace of an entity Person would be modelled as an attribute of type *entity* with target='Place':

```
<entity name="Person">
   <attributes>
      <attribute name="birthplace" datatype="entity" target="Place"/>
   </attributes>
</entity>
```

Section **types** provides the possibility of defining new types of datatypes: select or tree. More specifically, *select* and *tree*. User-defined types must be identified by a unique name to be used in the type attribute of the attributes using that type. In Table 4.7 a full toy example is reported.

attribute ... datatype=“select” tag
values
{**value**}

Table 4.5: Structure for describing enumerative attribute values in GBS files.

(**) (**attribute ... datatype=“tree”** | **values**) tag
values
{**value**} // see **(**)** (recursive)

Table 4.6: Structure for describing enumerative attribute values in GBS files.

As a conventional notation, we propose identifiers made up of uppercase letters, lowercase letters or decimal digits only. They should start with an uppercase letter for entity names and enumeration or tree values, or with a lowercase letter for domain, relationship and attribute names. Multi-word names are built by juxtaposing their constituent words, using an uppercase letter for the first letter of each word (except for the first one, as prescribed above). When writing documentation, a relationship ‘rel’ between an entity ‘Subj’ and an entity ‘Obj’ can be represented using the dot notation

Subj.rel.Obj

which is not ambiguous since dots are not allowed in our entity and relationship names.

Tables 4.7 and 4.8 show a fragment of a GBS file concerning the domain of computing. More specifically, you will find the “import” (*) section, for importing entities, relationships, and their respective attributes from another schema. Then, “types” specify specific user-defined types. In the example, it is a recurrent select datatype for which it makes sense to create an abstraction. Besides, “entities” lists all entities and their features. The hierarchy is shown. Finally, “relationships” mentions all the relationships with the corresponding “references”. We call “reference” a pair of (subject, object) of a relationship. They are intended as domain and range in the SW domain. Among the entities, we see “Component”, representing an electronic component and including a taxonomy of sub-classes. Moving down through the taxonomy, there is the class “Gate” whose function has been specified as a “logisticoperator” type, which is a user-defined type in the types (X) section. In 4.8, we see that the relationship “mayReplace” may

be established between a “Component” and another “Component”. This relationship has a child that is “replace”.

```
<domain name="retrocomputing" author="stefano" version="1">
    <imports>
        <import schema="general" />
    </imports>
    <types>
        <type name="logicalOperator" datatype="select">
            <taxonomy>
                <value name="NAND"/>
                <value name="AND"/>
                <value name="XOR"/>
            </taxonomy>
        </type>
    </types>
    <entities>
        <entity name="Award">
            <attributes>
                <attribute name="name" mandatory="true" distinguishing="true" datatype="string"/>
                <attribute name="date" mandatory="false" datatype="date"/>
                <attribute name="description"
                mandatory="false" datatype="text"/>
            </attributes>
            <taxonomy>
                <value name="Education"/>
                [...]
            </taxonomy>
        </entity>
        <entity name="Component">
        [...]
            <taxonomy>
                    [...]
                    <value name="Chip">
                        <taxonomy>
                            <value name="Logic">
                            [...]
                            <value name="Decoder"/>
                            [...]
                            <value name="Gate">
                                <attributes>
                                    <attribute name="function" mandatory="false"  datatype="logicalOperator"/>
                                </attributes>
                            </value>
                            [...]
            </taxonomy>
            [...]
        </entity>
        [...]
    </entities>
```

Table 4.7: Sample fragment of ontology in GBS format (part 1).

```
<relationships>
    <relationship name="acquired" inverse="acquiredBy">
        <references>
            <reference subject="Organization" object="Organization"/>
        </references>
        <attributes>
            <attribute name="date" mandatory="false" datatype="date"/>
            <attribute name="originalPrice" mandatory="false" datatype="real"/>
        </attributes>
    </relationship>
    [...]
    <relationship name="mayReplace" inverse="mayBeReplacedBy">
        <references>
            <reference subject="Component" object="Component"/>
        </references>
        <taxonomy>
            <value name="replace" inverse="isReplaced"/>
        </taxonomy>
    </relationship>
    [...]
</relationships>
```

Table 4.8: Sample fragment of ontology in GBS format (part 2).

The combination of many schemas is a more powerful representation than the simple juxtaposition of their elements. Indeed, their shared entities act as bridges that allow, through the relationships available in those domains, to connect proprietary entities of each domain that would not otherwise have a chance to be related to each other. In the GBS framework, classes and relationships in different ontologies are considered to be the same (and thus are shared) if they have the same name. They may have, however, different attributes, reflecting the different perspectives associated with the different domains. If an attribute is present in different domains it must have the same type in all of them. GBS schemas support this opportunity by providing an optional section in which existing schemas can be imported. The tag attributes are:

- **import** tag:

  schema: the name of a schema to be imported

- **delete** tag:

  elementtype = (entity | relationship)

  elementname: the name of the element to be deleted

Schemas are imported in the same order as specified by the sequence of **import** tags. Definitions of elements (entities or relationships) in an imported schema having the same name as elements defined in previously imported schemas merge the previous definitions. Finally, elements defined in the **entities** or **relationships** sections of the importing schema merge elements with the same name in all imported schemas. **delete**

tags allow to remove them from the overall ontology elements of the imported schemas that are not needed in the current domain.

They can be expressed through the tags, respectively, “functional”, “transitive”, “reflexive”. The last one covers the last two cases because reflexive=‘true’ specifies that it is while reflexive=‘false’ means that it is irreflexive. Symmetry is implicit in the name of the inverse.

4.4.3 Merging ontologies

Two ontologies may share the same concept, or describe the same concept in different ways or at different levels of granularity. In these cases, we allow the merging of classes and relationships (whenever possible) with the same name. Merging is a complex issue. Let’s consider a class hierarchy in which class A is a subclass of class B, be to merge with another one in which class B is a subclass of A. This cannot be resolved automatically because the solution might depend on the context of use, the general perspective behind those concepts, and the pragmatical reasons for which a designer can prefer one over the other.
This was just an example but there are many different similar situations impossible to be managed by a machine.
Let’s now describe the solution we adopted for the merging phase.

```
Input: Hierarchy H, Class C
Output: Hierarchy H

readTaxonomy(H, C):
    E = find(C, H)  // find if C is under H
    if E is null:
        E = find(C, ROOT)   // find C in the complete hierarchy
        if E is not null:
            if E is topClass:
                parent(E) = H
            else:
                if H is subClass of parent(E):
```

```
                parent(E) = H
            else:
                inconsistency
        else:
            create class E
            parent(E) = H
    else:
        concat_attributes(C)    // read attributes to add to C
        for child in C:
            readTaxonomy(C, child)
```

4.4.4 Mapping GBS and LPG

Data Representation Here we are going to describe how to map GBS elements into the formalism of LPG. For each concept (expressed through tag), we have a corresponding element in the LPG graph.

Instances of an entity expressed through the tag **entity** are represented as nodes, having as label the most specific entity they belong to. **Attributes** of entities may be filled or not. Whenever they are present, they are represented as properties of the specific node in the graph. This holds when attributes are of primitive types (string, integer, ...). Attributes of type "entity" are modelled as relationships, where the target entity represents the label of the object node of the relationship. This representation is fully transparent to the end-user which sees that particular element as an attribute in the schema (and interface).

Relationship is treated as an arc between two nodes, where the labels of subject and object are filled with the above-mentioned strategy. For the **attributes** of relationships, things change considerably. Primitive attributes are stored as properties of arcs. LPG supports properties on relationships as well. Concerning attributes of type "entity" or "date" we use reification that consists in creating a node labelled with the name of the relationships. Attributes of relationship. Since it is in lowercase, it will be self-evident that the node represents a relationship "instance" and not an entity instance. The latter are stored with the label in uppercase. To distinguish the reified node, the "subject" and "object" arcs are created from the reified node to the original

subject and object respectively. The interface will make this process transparent for the entities.

All in all, the graph contains only instances, not the schema, according to the following strategy:

- **node label**: the entity of an instance is represented as the label of the nodes. As a design decision, we reported only the most specific subclass of the node. Its superclasses can be derived from the schema specification.
- **property value**: attributes of classes and relationships are stored as properties. The LPG support attributes on both nodes and arcs.
- **arc type**: relationships are modelled as arcs in the graph. Sub-relationships are modelled in the same way.
- **user-defined type**: it is represented in the same way as attributes are. User-defined types are just an alias of attributes.

However, some exceptions exist due to the abstraction layer we introduced in GBS. In particular, they concern the following types of attributes:

- **entity**: the attribute is stored as a relationship. The interpretation of an attribute having as range a "target" entity is to be interpreted as a functional relationship to the instance of that entity.
- **date**: in the graph, dates are modelled as nodes of type *TimeSpecification* as already described.

With respect to the current state-of-the-art in the graph schemas definitions, I can compare GraphBRAIN Schema with the rest of the schemas in a table taken from [205]. The main features shown in Table 4.9 can be grouped into:

- **type features**: (PDT) the number of built-in primitive data types, (UIT) type constructors for union and intersection types, (TH) type hierarchies, (AT) abstract types, (OCT) open and closed types, (EP) edge properties, (MOP) mandatory and optional properties, (CPT) complex nested property types consisting of nested collection types, and (RC) range constraints.

- **constraint features**: (KC) key constraints, (MP) mandatory participation of certain types of nodes in certain types of edges, (CC) cardinality constraints for such participation, and (BRC) properties of binary relations defined by certain edges, such as (ir)reflexivity, (in)transitivity, (a)cyclicity, (a/anti)symmetry, etc.

- **schema features**: (TV) if validation is tractable, (ISP) if introspection is possible, i.e., the schema can be queried like a graph instance, and finally (SFPX) if the schema can be specified to be (1) first, and subsequently enforcing it for all instances, (2) partial, allowing some of its components to be descriptive (e.g., the element types) and some to be prescriptive (e.g., the constraints), or (3) flexible, creating and updating instances in an unconstrained manner while possibly maintaining a descriptive schema.

	PDT	UIT	TH	AT	OCT	EP	MOP	CPT	RC	KC	MP	CC	BRC	TV	IS	SFPX
Chen ER [226]	[-]	-	-	-	c	n/e	m	-	[-]	[✓]	✓	-	-	[-]	-	f
Extended ER [227]	[-]	n/e	n/e/p	n/e	c	n/e	m/o	✓	[-]	✓	✓	✓	-	[-]	-	f
Enhanced ER [228]	[-]	n/e	n/e	n	c	n/e	m/o	✓	[-]	[✓]	✓	✓	-	[-]	-	f
ORM2 [229]	[-]	n/e/p	n/e/p	n/e	c	n/e	m/o	[-]	✓	✓	✓	✓	✓	[-]	-	f
UML Diagrams [230]	[5]	n	n	n	c	n/e	m/o	[-]	✓	✓	✓	✓	[✓]	[-]	-	f
RDFS [231]	34	-	n/e/p	-	o	[-]	[o]	-	-	-	-	-	-	✓	✓	f/[p]/x
OWL [232]	[33]	n/e/p	n/e/p	-	o	n	[m]/[o]	-	✓	✓	[-]	✓	✓	[✓]	✓	f/[p]/x
SHACL [208]	34	n/e/p	n/e/p	-	o/c	n	m/o	-	✓	[✓]	✓	✓	-	[✓]	✓	f/p/x
ShEx [185]	34	n/e/p	n/e/p	-	o/c	n	m/o	-	✓	[✓]	✓	✓	-	[✓]	✓	f/p/x
DTD [233]	6	[n]	-	-	o/c	n	m/o	[-]	-	[-]	✓	-	-	✓	[✓]	f/x
JSON Schema [234]	6	n/e	n/e/p	n	o/c	n	m/o	✓	✓	[✓]	✓	✓	-	✓	✓	f/x
RELAX NG [235]	[2]	n	n/e/p	n	o/c	n	m/o	✓	✓	[-]	✓	-	-	✓	✓	f/x
XML Schema [236]	[47]	n	n/e/p	n	o/c	n	m/o	✓	✓	✓	✓	✓	-	✓	✓	f/x
GraphQL SDL [207]	5	n/[e]	n/e	n/e	c	n/e	m/o	✓	[✓]	✓	✓	-	-	[-]	✓	f/[p]
openCypher [237]	[oC]	[n]	n/e	n/e	c	n/e	m	✓	-	✓	✓	-	-	✓	✓	p/x
SQL/PGQ [238]	[SQL]	-	[n]/[e]	n/e	c	n/e	m/o	✓	[✓]	✓	✓	[-]	-	✓	-	f
GQL [239]	[-]	-	-	-	[o]/c	n/e	m/o	-	-	-	-	-	-	✓	-	f/x
AgensGraph [240]	[-]	-	n/e/p	-	o	[n]/[e]	m/o	[✓]	[✓]	[✓]	✓	-	-	[✓]	[✓]	f/[p]/x
ArangoDB [241]	6	[n]/[e]	n/e/p	n	o/c	n/e	m/o	✓	[✓]	[✓]	✓	[✓]	-	[✓]	✓	f/x
DataStax [242]	[25]	-	-	-	[o]	n/e	m/o	✓	[✓]	✓	✓	-	-	?	✓	f/[p]/x
JanusGraph [243]	12	-	-	-	[o]	n/e	m/o	✓	-	✓	✓	✓	-	✓	[✓]	f/[p]/x
Nebula Graph [244]	5	-	-	-	c	n/e	m/o	-	-	-	-	-	-	[-]	✓	f
Neo4j [51]	11	[n]	[-]	[-]	o	n/e	m/o	[✓]	-	✓	-	[-]	-	-	✓	p/x
Oracle/PGQL [245]	11	[n]/[e]	-	-	c	n/e	[m]/[o]	-	[✓]	[✓]	[✓]	[-]	-	[✓]	✓	f/[p]/x
OrientDB/SQL [246]	23	[n]/[e]	[n]/[e]	[n]/[e]	o/c	n/e	[m]/[o]	✓	[✓]	[✓]	[✓]	[-]	-	[✓]	[✓]	f/[p]
Sparksee [247]	8	-	-	-	c	n/e	m/o	[✓]	-	✓	✓	-	-	[-]	[✓]	f
TigerGraph/GSQL [248]	8	[n]/[e]	-	[-]	c	n/e	[m]/[o]	✓	[✓]	✓	-	[-]	[✓]	[✓]	✓	f
TypeDB/TypeQL [249]	5	-	n/e/p	n/e/p	c	n/e	m	✓	[✓]	✓	✓	[-]	✓	[-]	✓	f
PG-ScHEMA [205]	[-]	n/e	n/e/[-]	n/e/[-]	o/c	n/e	m/o	[-]	[-]	✓	✓	[-]	[-]	✓	-	f/p/x
GBS Schema	**[3]**	**[-]**	**n/e**	**n**	**o**	**n/e**	**m/o**	**[-]**	**[-]**	✓	✓	**[-]**	**[-]**	✓	**-**	**f**

Table 4.9: '✓': supported, '-': not supported, '?': unknown, [x]: *qualified* x, n/e/p: supported for (n)odes, (e)dges, and (p)properties, o/c: (o)pen and (c)losed, m/o: (m)andatory and (o)ptional, f/p/x: schema (f)irst, (p)artial, and fle(x)ible, oC: openCypher

Chapter 5

Methodology for GB-based Solutions

This chapter describes the results of the applications of GraphBRAIN technology to solve relevant problems in several domains of interest. First, we describe what we refer to as "holistic" conceptualizations, which merge domain-specific entities with other more general real-world entities. We do not intend to propose *finished* schemas, since we are aware from history that it is not practically feasible. What is proposed here are extended conceptualizations for existing domains, in which we analyzed the main lacks and possibilities for extensions. In this realm, I show the benefits of these conceptualizations, how to use them, and propose modern solutions that surpass previous limitations.

5.1 Independent Learning

In a world evolving at an incredibly fast pace, there is an ever-growing need for continuous training and education. In traditional education, most teachers cannot give every single student in the class the time and attention needed to overcome his individual difficulties or to leverage his particular interests. E-learning takes this problem to the extreme, expanding the audience from the dozens to the thousands, which makes it hopeless to have a single method or material fit the needs of all users. On the other hand, adaptive learning aims at addressing the specificities and differences between individual learners. This should make them more comfortable in the learning effort and allow them to understand and master the concepts more efficiently [250, 251]. Intelligent Tutoring Systems (ITSs) represent the solution. These systems should foster

the learner's engagement and motivation since they are intended to: recognize relevant skills concerning career goals, find topics of interest, retrieve missing blocks of knowledge and control the learning journey. If new interests emerge during a learner's experience, the system might expand recommendations in that direction or change pathways accordingly. Regardless, it will search for engaging educational experiences. Here we propose a logical architecture for KEPLAIR (Knowledge-based Environment for Personalised Learning using an Artificial Intelligence Recommender) [252], an ITS designed to embody such a vision. Using this tool, learners can live educational experiences tailored to their educational goals, prior knowledge, personal interests and preferences, abilities, and contexts. The system will pervasively use AI to carry out its tasks. Among them, a crucial role is played by an ontology, that informs all the internal representations and behaviour, so as to smoothly connect and orchestrate all the various functions and ensure both internal and external interoperability.

5.1.1 KEPLAIR's Architecture

The logical architecture of KEPLAIR is shown in Fig. 5.1. Since KEPLAIR is an intelligent agent, some components (SeaL, ReaL and LeaL) are inspired by the architecture proposed in [253] for agents in Ambient Intelligence applications. The cloud on the top represents the Internet, from where users can connect to KEPLAIR and many Learning Objects (LOs) reside. Users interact with KEPLAIR using the SeaL (Sensors, Effectors and Applications Layer) component, which includes all application interfaces and sensors and effectors available in the environment. The main system is in the middle layer, which coordinates all processing tasks and processes involved in providing educational services by pervasively exploiting an underlying AI level that provides all the intelligent functions and serves the user interfaces. It consists of three components, each including several specific modules to carry out its function:

- the Harvesting Manager is in charge of identifying and collecting the LOs and their associated metadata.

- the Learning Manager simulates human interactions via AI, acting as an automatic tutor, counsellor, and personalised assistant to build learning paths, recommend LOs, build tests to check their performance, etc.

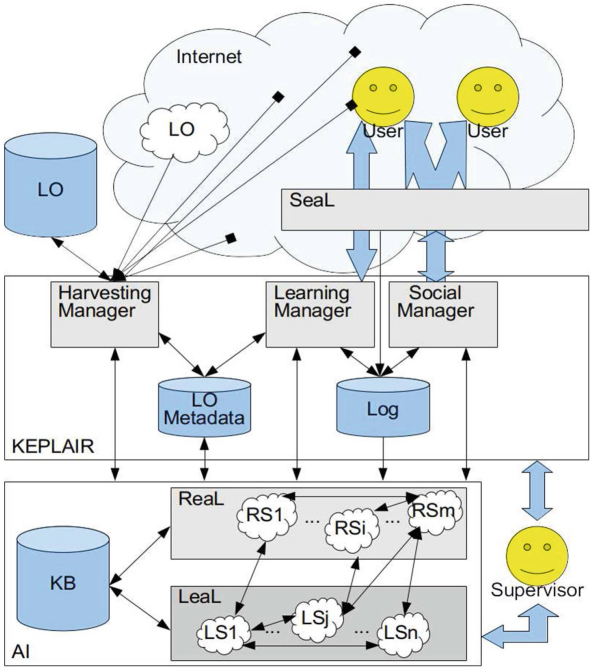


Figure 5.1: KEPLAIR's Architecture

- the Social Manager handles interactions among users and communities of users in the educational context, possibly intervening to provide support.

All the knowledge useful for KEPLAIR's operation is stored in a KB, including the users'profiles (built from both information explicitly provided by the users and information automatically extracted from the Log repository). It is conceived as a KG. To obtain both efficient data handling and effective knowledge manipulation, it is implemented using the GraphBRAIN technology [222]. Data are stored in a graph DBMS, and their schema is stored in the form of a comprehensive ontology, possibly obtained by merging and aligning several domain-specific ontologies. The ontology is a fundamental reference for the entire system, coordinating all components and modules during their information exchanges and ensuring they are interoperable and assign the same meaning to the same pieces of data.

5.1.2 KEPLAIR's Ontology

In designing the ontology for KEPLAIR we started from existing schemas and ontologies used in the Digital Libraries and education fields, trying to merge them into a coherent compound schema. Specifically, for the educational domain, our starting point was the set of ontologies used in the IntelLEO European project[1], including

[1]https://cordis.europa.eu/project/id/231590

ALOCOM [80] and SCORM[2]. We also took into account the Learning Object Metadata (LOM) standard [254]. For the digital library domain, useful to describe documents and learning materials, we considered the Functional Requirements for Bibliographic Records (FRBR) of the International Federation of Library Associations and Institutions (IFLA)[3], the Dublin Core Metadata Initiative (DCMI)[4], the METS[5] and the Open Archives Initiative Object Reuse and Exchange (OAI-ORE) standards[6] for the description of compound objects. We further expanded the resulting ontology by adding classes, attributes and relationships to account for the content and context information that is often neglected in the state-of-the-art, especially concerning the specificities of each individual student. It includes several general classes relevant to the education domain, among which are **User**, which represents the concept of the learner that uses the system to earn new knowledge; **Person**, which represents any possible relevant person for users. **Place**, with its several possible subclasses. **Environment** describes the different kinds of environments in which users can be immersed. **Activity** represents the set of tasks to be performed to reach any kind of goal. Other useful classes are **Organization** and **Event**. These classes are borrowed from an ontology of general, widely used concepts, reusable in most domains. Then, there are the domain-specific classes for education. The most prominent kinds of LOs are **Document**s and its several subclasses at different levels (the first one distinguishing **Printable**, **Audio** and **Video**). **AssessmentTool** represents tools that can be used to check if users possess given skills (e.g., tests). Other classes are **Container** (to describe classes, courses, etc.) and **AccomplishmentEvidence** collects certifications (micro badges, certificates, licenses, standardised tests, degrees, diplomas) stating that a user possesses given skills, **Artifact** (including **Handicraft**s, **IndustrialWork**s, and **Artwork**s, such as statues and paintings), **Device** (including simple tools, such as hammers, and more complex systems, such as computers), **Software** (again, with a taxonomy of sub-classes for the different kinds of software). For fine-grained handling of LOs (and especially documents), KEPLAIR's ontology also allows for the description of their structure and content. **DocumentDescription** allows for the description of documents' layout and

[2]https://scorm.com/

[3]https://repository.ifla.org/server/api/core/bitstreams/42b598cb-9a1f-4ede-be6d-468bc0f4e9bc/content

[4]https://www.dublincore.org/

[5]https://www.loc.gov/standards/mets/

[6]https://www.openarchives.org/ore/

logical structure, their text and their grammatical structure (**Sentence**, **Subject**, **Object**, ...). KEPLAIR provides many other relationships to connect instances of these classes. Some are needed to organize documents into a layout and logical components (e.g., **has** links a **Document** to its layout and logical components; **partOf** structures components into sub-components, etc.) Layout components also have a spatial organization, expressed by relationships **leftOf** and **above**. Authors of documents and LOs are connected to them through the **developed** relationship. Due to lack of space, I will not mention all the other relationships, most of which can be easily guessed by the readers, those linking **Person**s to **Organization**s, or those describing the syntactic and semantic structure of the text, and so on.

5.1.3 Learning Materials Recommendation

With reference to KEPLAIR's architecture, the recommendation engine is located in ReaL, and draws from the KB the knowledge necessary to carry out its task. The recommendation function currently embedded in KEPLAIR leverages the information expressed in the form of descriptors associated with the items (users, courses, materials) in the KB. The tags may have been obtained by previous knowledge. The recommendation strategy is based on a mix of different kinds of computation, as proposed in [255]: graph mining and network analytics are exploited to obtain the relevant information from the KB; reasoning is used to drive the graph navigation and exploit high-level knowledge that is not explicitly expressed by the instances, and finally numeric computation is used to compute relevance scores needed to filter and rank recommendations. One of the tasks is a material recommendation. For this task, we have for each user:

- the courses he/she has already attended, in the case of course recommendation.
- the materials belonging to the current course, plus the materials that the user has already exploited, in the case of learning material recommendation.

From the extracted subgraph, only nodes corresponding to courses (resp., learning materials) are extracted as candidates, and the courses (resp., learning materials) already exploited by the user are removed.

We base our similarity assessment on the distance between nodes (tags) in the graph, expressed as the number of arcs in a path connecting them (specifically, we

consider the shortest paths) in Table 5.1. The distance of a node from itself is 0; it is undefined when there is no path connecting those nodes in the graph.

Table 5.1: Fragments of subjects' taxonomy

Computer Science	**Medicine**	**Mathematics**	**History**
Database	Anatomy	Statistics	Roman history
SQL	Tissue	Mean	Roman Empire
Relational Model	Articulation	Variance	
NoSQL	Blood	Median	
Graph			
Programming	Pathology	Analysis	Modern history
C	Virus	Line	French Revolution
Java	Bacteria	Parabola	
Python	Toxins	Hyperbola	
AI	Pharmacology	Geometry	Contemporary History
Machine Learning	Antibiotics	Triangle	World War II
Neural Networks	Antihistamines	Square	
Expert System	Anti-inflammatories	Cube	
Algorithm			
Complexity			
Network			
Protocol			
Security			
Cryptography			

Items are described by two sets of tags. We base the similarity assessment between sets of tags on a modified version of the Jaccard index, a simple but widely used [256, 257] formula for assessing the similarity between two sets. In our case, however, the set members are not independent in general. They may be directly or indirectly related to the KB. For this reason, instead of set operations, we leveraged the notion of distance and computed the ratio between the sum of the (defined) distances between pairs of related tags and the number of pairs having a defined distance. The resulting value is used as a parameter in the following *similarity* function s:

$$s(x) = \frac{1}{a^x}$$

for some $a > 0$. Independently of a, we have:

$$x = 0 \implies a^x = 1$$

So far, non-related pairs (having undefined distance) have not played any role. To

take them into account, we further computed a penalty to be used to smooth the score:

$$P = 1 - r^b,$$

where r is the ratio of non-related pairs of tags in the two sets over the total number of tags, while b is a parameter that affects the weight of the penalty. When all pairs of tags are related, $r = 0 \implies P = 1$, so no penalty is applied to the score. The whole solution was implemented in Prolog where the most relevant facts are reported:

- **student(X)** : X is the username of a registered user
- **course(X,Y)** : Y is the name of the course having id X
- **learningObject(X,Y)** : Y is the name of the LO having id X
- **learningObjectTag(X,Y)** : LO X is described by tag Y
- **hasLearningObject(X,Y)** : course X includes LO Y
- **hasRead(X,Y)** : user X used LO Y
- **userHasReadTag(X,Y)** : returns the tags of an object seen by the user.
- **taglist(X,Y)** : returns all tags describing an item.
- **distance(X,Y,D)** : returns the distance between two nodes, if it exists, or undefined if they are not connected in the graph.
- **distanceTaglistTaglist(L1,L2,D)** : returns the list of all pair-wise distances between the tags in two lists.
- **scoreFunction(X,S)** : computes the score function seen by the user and all unseen objects.
- **orderedScoreUserObject(X,O,S)** : returns all objects to be recommended to the user, ranked by score.

5.1.4 Experiment

To carry out a preliminary test of our recommendation approaches, we populated KEPLAIR's KG with sample data. As to the topics, we considered the domains of computer science, mathematics, humanities and medicine. We also added information about a few test learners. E.g., a portion of the information associated with user 'davide', expressed as Prolog facts used by the recommendation engine. Let us show a use case in this setting, concerning user 'davide'. Applying the methodology described above to recommend new lectures to 'davide', based on the lectures he has already attended. Specifically, 'davide' attended lectures about Database, Programming, AI, Programming Languages, History, Computer Networks, Data Mining. Using parameter values a = 3 and b = 0.5 for the score computation, here is the set of lectures recommended to 'davide', ranked by decreasing score. With reference to the Prolog engine, they were obtained running query:

$?- orderedScoreUserObject('davide', X, \_).$

having as results:

$X =' Lecture3 - ComputerNetworks';$

$X =' Lecture1 - Cybersecurity';$

$X =' Lecture2 - Cybersecurity';$

$X =' Lecture4 - IA';$

$X =' Lecture1 - AlgorithmsandDataStructures';$

$X =' Lecture3 - IA';$

$X =' Lecture3 - ProgrammingLanguages';$

$X =' Lecture3 - AlgorithmsandDataStructures';$

$X =' Lecture5 - Programming';$

$X =' Lecture5 - IA';$

$X =' Lecture4 - Programming';$

$X =' Lecture4 - Database';$

$X =' Lecture3 - Programming';$

$X =' Lecture3 - Database';$

$X =' Lecture2 - Programming';$

$X =' Lecture2 - Database';$

$X =' Lecture2 - ComputerNetworks';$

$X =' Lecture3 - DataMining'$;

$X =' Lecture3 - Cybersecurity'$;

$X =' Lecture3 - History'$;

$X =' Lecture2 - History'$;

$X =' Lecture2 - ProgrammingLanguages'$;

$X =' Lecture2 - AlgorithmsandDataStructures'$;

$X =' Lecture2 - IA'$;

$X =' Lecture2 - DataMining'$;

$X =' Lecture4 - Statistics'$;

...

$X =' Lecture3 - Geometry'$;

Even if we do not go into the numerical values dictated by the calculation of similarities, we can guess the reasons behind why some LOs are suggested before others. The first is "Lecture 3 - Computer Networks", which, as we can see, presents the tags "computer science", "computer networks" and "algorithm" which are all present among the tags of the LOs that the user has already visited. The next two LOs, on the other hand, also contain the tag "cybersecurity", which is not present among the tags of the LOs seen. The order, however, is not indifferent. One of the LOs has the tags "computer science", "computer networks" and "protocols". The second and third suggested LOs have the same tags except for the last one, as "computer networks" appear in the second and "protocols" in the third. Since the tags are placed in order of specificity, more importance is given to "computer networks" since it appears as the second tag, and therefore the second LO takes precedence over the third. Finally, proceeding rapidly downwards, we find more and more LOs with fewer (relevant) tags in common with those present in the LOs visited. The last suggested LO does not contain any tags present among the tags of the visited LOs but contains "statistics" which is related to the LO of the IA lesson. After running a few experiments and asking the sample users to rate the recommendations, we may preliminarily conclude that the set of suggested Learning Objects was generally considered satisfactory and useful. Since the recommended lectures are ordered by relevance, the users confirmed that the very first LOs are more closely related to what they had already studied, and represent good paths to expand their knowledge in those directions. Experiments showed that the top 5 LOs are highly correlated to user needs, and information about precedence (e.g. "IA" must

follow the course of "Algorithms" may easily be injected by logic rules).

5.1.5 Evaluation

I can evaluate this solution with respect to a family of previous works. Gasmi et al. [258] introduced OWL schemas to formalize connections between profiles and competencies (which can be seen as the equivalent of LOs), and with Semantic Web Rule Language (SWRL) [259] specify how to correlate the two areas. This work provides good results for the task it aimed to solve, but it was likely impossible to generalize into another domain. The time needed to introduce SWRL rules is of considerable size. Moreover, OWL rules are monotonic and work under the Open World Assumption (OWA). With a Prolog-based formalization, we can deal with both of these features, and no rule-specification process is needed. At least the problem of monotonicity of SWRL rules has been tackled by Fortineau et al. [260] but with performance and expressiveness problems related to how to automatically guide the reasoning process introducing all the possible rules to "close" the world.

It is becoming more and more frequent to combine symbolic approaches with Machine Learning (ML) as in Villegas et al. [261]. Although this approach seems the most general, reliable, and performing, it needs a quantity of data that is not always guaranteed for a specific use case or domain.

5.2 Digital Libraries

The traditional record-based approaches to library information description are nowadays insufficient and unable to fully grasp and express the complexity of such information [262, 263], both internally and externally. Internally, the main limitations (deriving from legacy paper cards) are that (i) a record is document-centric, reporting a number of information elements all considered as "belonging" to the described document and (ii) it includes a small, pre-defined and fixed, set of fields. Externally, the strictly sequential organization of records deriving from legacy files makes no more sense for computer files. Another limitation is the lack of semantics. Even most recent proposals in this direction mostly deal with the description level, without making the leap to semantic technologies, driven by ontologies, that alone can open new landscapes and

enable advanced AI support to the field. Concerning internal limitations, it is necessary to distinguish the descriptive elements that are inherent in the document from those that exist on their own, and just "happen" to be associated with the document for some reason. Also, the set of descriptors must become much larger, varied and flexible, allowing to connect the library items to all the wealth of knowledge directly or even indirectly related to them but precious, or sometimes crucial, for practitioners, researchers and end-users to understand the items properly. In my vision, this includes the content of the documents, their materiality, their context and their lifecycle (involving their users and their uses). Even information not connected to the traditional library descriptions might be used to indirectly connect the documents and, ultimately, provide a deep understanding thereof. As to external limitations, an upgrade to a reticular, graph-based approach, is required, where the different kinds of entities involved in such knowledge live on their own, rather than being just values in record fields, and where relationships among entities are the key feature of descriptions. This is also instrumental to enabling semantic technologies, typically based on graph structures.

5.2.1 IFLA-based Ontology

I describe here the development of the ontology. I focused on the portion of holistic ontology that corresponds (i.e., is aligned and compliant) to the IFLA reference models FRBR and LRM. More specifically, due to space constraints, we focus here on the entities, that are central and preliminary to determining the relationships and attributes. The resulting alignment of entities is reported in Table 5.2. In order to provide a slightly broader view of the ontology, it also reports GraphBRAIN entities that are superclasses of classes aligned with IFLA entities but have no counterpart in IFLA standards. The core of the ontology must concern established, necessary or useful descriptors provided for by library and archival theory, standards or practice. For entities, I considered mandatorily that the ontology includes all the items in the FRBR and LRM conceptual models proposed by IFLA to make up the core portion of the class hierarchy. A preliminary alignment was required between the IFLA standards, FRBR and LRM, as well since they sometimes use different names for the same concepts or give different interpretations to concepts with the same name. For instance, *Agent* in LRM corresponds to *Responsible Entity* in FRBR (where the name reflects a specific

focus on the role), which were mapped onto the **Agent** entity in GraphBRAIN. The strategy for identifying subclasses, specifically under **Expression** and **Document**, was to define specific subclasses for those attributes in which the IFLA proposals specified applicability to some specific kinds of documents only.

GraphBRAIN	**FRBR**	**LRM**
Entity	-	-
+ Category	-	-
+ + Concept	Concept	-
+ + Object	Object	-
+ Agent	Responsible Entity	Agent
+ + Person	Person	Person
+ + Organization	Corporate Body	Corporate Body
+ Intellectual Work	-	-
+ + WorkOfArt	Work	Work
+ + + MusicalWork	Work	-
+ + + CartographicWork	Work	-
+ + Expression	Expression	Expression
+ + + SoundExpression	Musical or Sound	-
+ + + + MusicExpression	Musical	-
+ + + CartographicExpression	Cartographic	-
+ + + RemoteSensingImage	Remote Sensing Image	-
+ + + VisualExpression	Graphic or Projected Image	-
+ Document	Manifestation	Manifestation
+ + Printable	-	-
+ + + PrintedBook	Printed Book	-
+ + + + HandPrintedBook	Hand Printed Book	-
+ + Audio	Sound Recording	-
+ + Visual	-	-
+ + + Image	Image	-
+ + + Microform, Projection	Microform or Visual Projection	-
+ + Computer File	Electronic Resource	-
+ + + RemoteComputerFile	Remote Access Electronic Resource	-
+ Item	Item	Item
+ Collection	-	-
+ + Series	Expression>Serial, Manifestation>Serial	-
+ Nomen	Nomen	Nomen
+ Collection(Word), Taxonomy	-	Schema
+ Place	Place	Place
+ Event	Event	-
+ TemporalSpecification	-	-
+ + TimeInterval	-	TimeSpan

Table 5.2: Comparison of FRBR and LRM Concepts

5.3 Diachronic Analysis

Diachronic analysis is one of the most intriguing and complex tasks in linguistic studies. The interest in the task is well supported by the research on historical languages such as Latin. The amount of data in this language is a great opportunity to uncover unknown semantic changes. Integrating lexical semantic information and diachronic language resources is essential in enabling quantitative accounts of language change. I present results from integrating Latin corpus data, Latin WordNet, and Wikidata into a graph database via the GraphBRAIN Schema and show the potential offered by this model for diachronic semantic research.

5.3.1 Dataset

LatinISE contains 10 million word tokens from texts dating from the fifth century BCE to the contemporary era; it has been semi-automatically lemmatized and part-of-speech tagging. The corpus includes metadata fields indicating text identifier, author, title, dates, century, genre, URL of the source, and book title/number and character names (for plays). The annotated dataset was created as part of the SemEval shared task on Unsupervised Lexical Semantic Change Detection [104] and will be henceforth referred to as the SemEval Latin dataset. It contains annotations for 40 Latin lemmas, 20 of which are known to have changed their meaning concerning Christianity (for example, "beatus", which shifted its meaning from "fortunate" to "blessed"), and 20 are known not to have changed their meaning between the BCE era and the CE era. For each of these lemmas, 60 sentences were annotated, 30 randomly extracted from BCE texts, and 30 from CE texts. The annotation was conducted following the DuReL framework [264], where the semantic relatedness of an instance of usage of a target word to the list of its dictionary definitions was annotated using a four-point scale (Unrelated, Distantly Related, Closely Related, and Identical). The definitions were from the Logeion online dictionary (https://logeion.uchicago.edu/) containing Lewis and Short's Latin-English Lexicon (1879) [265], Lewis' Elementary Latin Dictionary (1890) [86]. The details of the annotation are described in McGillivray et al. [106].

5.3.2 Linking

We manually linked each word sense of the SemEval Latin dataset to one or more WordNet synsets. We started with the dataset provided by the LiLa project [266], which contains a sub-sample of 10,314 lemmas from Latin WordNet (LWN) [102, 103]. The LiLa team verified and corrected, where necessary, the synsets associated with each lemma of the sub-sample and linked them to version 3.0 of Princeton WordNet (PWN) [223, 267]. However, as the LiLa dataset only covers 22 of the 40 lemmas in our dataset, we used Short's LWN as a reference for the remaining 18 lemmas. We converted the synset codes 1.6 used by Short's LWN to version 3.0 of PWN for consistency. The senses assigned to the target words in the SemEval Latin dataset often condensed multiple meanings into a single definition, requiring multiple synsets to be linked to the same meaning to capture all nuances. For example, with *consilium* (a lemma in the LiLa dataset), the sense "understanding, judgment, wisdom, sense, penetration, prudence" required four synsets to represent it effectively. In some cases, none of the assigned synsets in the LiLa dataset could describe a particular sense. In such cases, we searched for the lemma in LWN and recovered a synset that LiLa had deleted. When we could not find the synset in either LWN or the LiLa dataset, we looked for the most suitable synset in PWN. However, for some meanings specific to Roman culture and institutions, we could not find a suitable synset, such as with *virtus*, and its meaning "virtue, personified as a deity". In these cases, we did not link the sense to WordNet.

In some instances, the metadata field of the SemEval Latin dataset (which indicates the author and title of the text, dating, and genre) was noisy, incorrectly structured, or incomplete. Wikidata is an extensive knowledge base [268] hosting more than one hundred million items collaborative maintained. We exploited Wikidata for denoising and linking the authors of the documents to which the sentence refers. First, we extracted the Wikidata entities for which the author's occupation is specified (wdt:P106, occupation), and Latin (wd:Q397, Latin) is one of the writing languages of the author (wdt:P6886, writing language). The term *occupation* is used in a broad sense, to refer to various types of profiles that identify authors in Wikidata. These could be e.g., priest, philosopher, historian, hagiographer, among others. For each author, we retrieve the information about him/her in the form of key/value properties. Author names in

the SemEval Latin dataset can occur in different languages and different forms, e.g., praenomen and nomen followed by cognomen e.g., Marcus Tullius Cicero; cognomen followed by praenomen and nomen e.g., Cicero, Marcus Tullius; only cognomen e.g., Cicero; only praenomen and nomen e.g., Marcus Tullius. We processed the author's mentions in the SemEval Latin dataset and the writer labels and aliases extracted from Wikidata, performing lowercase and punctuation removal. Matching is realized by computing the Levenshtein distance [269] between the author reported in the SemEval dataset and all the collected surface forms (i.e., labels/aliases) from Wikidata. According to Levenshtein distance, the surface forms are then ranked in decreasing order. If the Levenshtein distance between the author's mention and the top-ranked surface form is less than a fixed threshold, i.e., $\delta = 0.1$, the entity referenced by the surface form is linked to the author's mention. For each author, Wikidata provides rich information, such as biographical data, the author's works, and events that influenced the life and the writer's production. In this study, we focus on the occupation information: we encode the information provided by Wikidata about the occupations of the author exploiting the property wdt:P106 (occupation). In particular, we create nodes of type **Occupation** for each occupation retrieved in Wikidata, generating a relationship between the author and their respective occupation.

5.3.3 Linguistic Ontology

To address the need to create a common shared vocabulary to visualize and connect data, we describe our linguistic ontology's main components. This schema collects all the relevant pieces of information available in standard lexical databases and other relevant sources of knowledge for diachronic analysis. From now on, we report the classes and relationships of our ontology in boldface; words are represented in lower-case, and relationships in upper-case. A central element is **Document**, which represents the hub for knowledge discovery in this field since it contains most aspects of knowledge that we need. It is linked with the Person who wrote it (*has_author*), commonly named the "author". A document may *concern* specific Artifacts, Devices, belong to (*belongs_to*) one Category, be written in at least one (*has_language*) Language and published in (*published_in*). We represent Texts belonging to *belongs_to*) documents. From the text, we are able to represent the **Word**s it contains. Lemmas are a specific type of

words, with their information, e.g., morphology, and part of speech tags. On the other hand, words have (*has_lemma*) lemmas. Synsets have relationships with each other; one may be a sub-synset of another (*is_a*) or be equivalent to (*same_as*) another one in another database. This happens naturally when mapping WordNet with Latin WordNet. Time needs to be modelled for diachronic analysis. **TemporalSpecification** includes **TimeIntervals** and specific **TimePoints**. The latter includes **Year**, **Month**, and **Day**. This model allows authors and manuscripts to be bound to specific time periods. Moreover, we have **Event**s, which may come in handy to understand the reason why some words changed their meaning (e.g., the advent of Christianity).

5.3.4 Latin WordNet Ingestion

The Latin WordNet (Latin WN) project is an initiative to create and share a common lexico-semantic database of the Latin language. The project is a branch of the MultiWordNet [270] project. For the application of diachronic analysis, linking linguistic resources with temporal information allows us to uncover instances of semantic changes in the usage of words. Hence, we provide a mechanism to enrich the Linguistic Knowledge Graph with Latin WordNet and exploit the hierarchical structure of the relationships among synsets. We map the Latin WordNet concepts with the portion of our ontology specifically devoted to linguistic analysis and understanding. Here we describe the mapping between the lexical database and our schema. In Latin WordNet, we identify the following resources, grouped into separate CSV files: lemma, lexical_relation, literal_sense, metaphoric_sense, metonymic_sense, phrase, semantic_relation, synset. Each resource has features that may be seen as classical columns in the relational databases perspective. From now on, we refer to specific fields as resource.field to uniquely identify them and motivate how we map them. The alignment process is as follows:

- lemma: a specific lemma is embedded in our **Lemma**. A lemma is characterized by a unique id, a lemma (its value), and a PoS tag (modeled as a relationship). For our purposes, the class **PartOfSpeech** collects all the pos tags used, following the Universal Pos Tags standard. We can represent other fields expressed in Latin WN, such as lemma.uri.

- lexical_relation: this represents a relationship between two lemmas. The field

lexical_relation.type specifies the type of relationship. We modelled the present ones with some explicit names which express their meanings: antonymousOf, pastParticipleOf, pertainsTo with their corresponding inverses.

- literal_sense: this represents a relationship between a lemma, identified by the field literal_sense.lemma, and a synset, identified by literal_sense.synset. We call this relationship expresses. We highlight that the relationship has a "literal" sense by adding a specific attribute sense. Additional information about the period and genre is available.

- metaphoric_sense: this relationship is expressed in the same way as the previous one, apart from the fact that the sense is "metaphoric".

- metonymic_sense: the same as before, but the sense is "metonymic" in this case.

- phrase: a phrase is a word or a multi-word expression. In both cases, the concept is expressed by the class Lemma. Again, we have the PoS tag information, which is modelled in the same way described above.

- semantic_relation: a relationship between two synsets. Based on the semantic_relation.type several relationships may be expressed. They are mapped into the following ones and their corresponding inverses: partOf, has_Subclass, attributeOf, similarTo, antonymousOf, pertainsTo, pastParticipleOf, causes, and entails.

- synset: a synset is mapped into a Lexicon-Concept. Matching the Latin synset with the synset in WordNet is possible by concatenating synset.pos and synset.offset. synset.gloss is the description of the synset and is mapped onto the attribute description.

Thanks to this mapping, we can acquire the Latin WN resource and represent it in our formalism, enabling us to leverage the connections between the different datasets, as explained via examples in the next section.

5.3.5 Semantic Change Analysis

Figure 5.2 shows the subgraph for the word *humanitas*. The occurrences of humanitas are annotated in the SemEval dataset with three senses: (i) 'human nature, humanity',

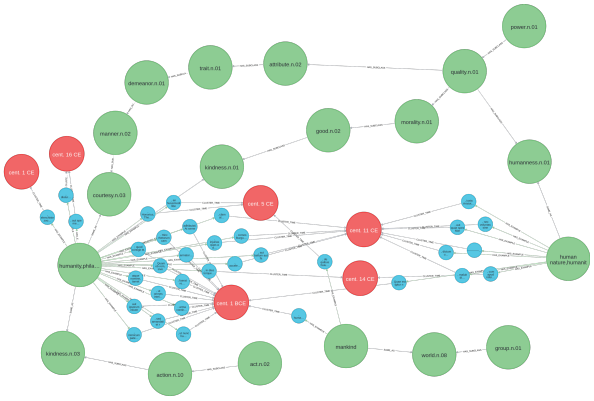
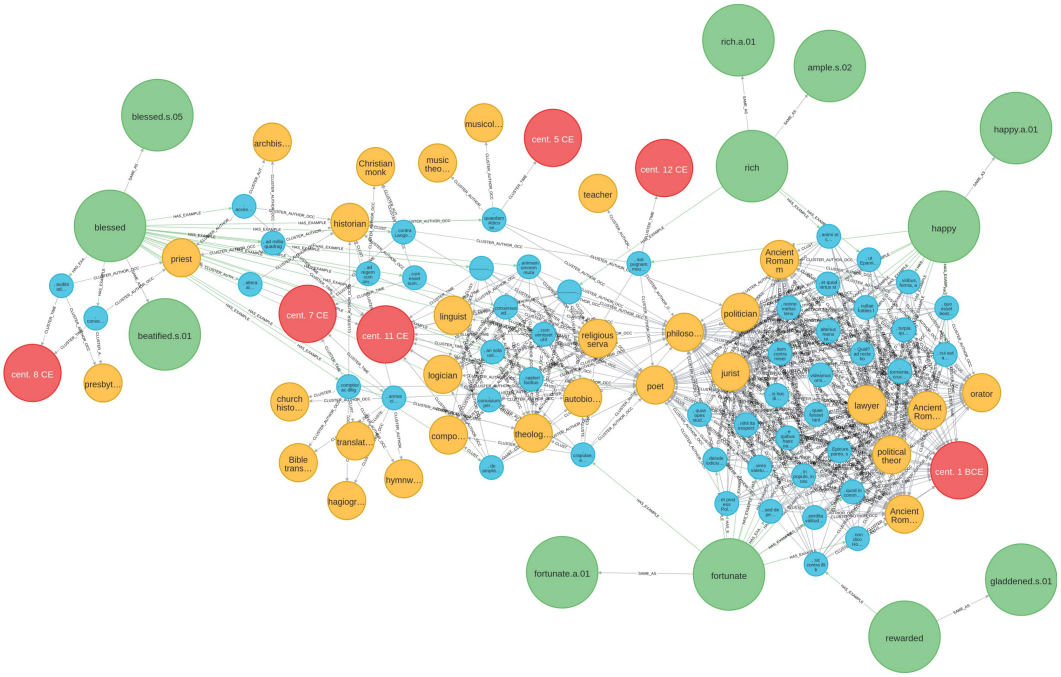


Figure 5.2: Subgraph for the word humanitas, including the sentences in which the lemma humanitas occurs in the SemEval Latin dataset, the century of the works from which the sentences were extracted, the annotated senses in the SemEval Latin dataset, and the curated links between the senses and the synsets in Latin WordNet. The sentences are represented as Text nodes (in blue), the senses and the synsets as LexiconConcept nodes (in green), and the centuries as TimePoint nodes (in red)

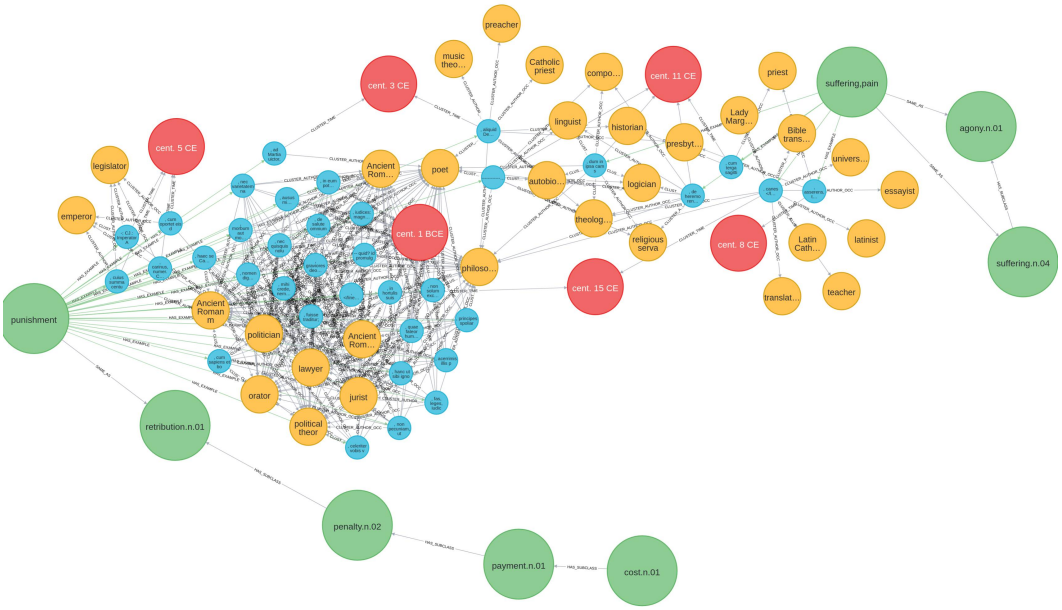
(ii) 'humanity, philanthropy', and (iii) 'mankind'.6 In the curated link, we associate the sense (i) to the humanness.n.01 synset, the sense (ii) to the synsets kindness.n.01, kindness.n.03, and courtesy.n.03 and sense (iii) to the synset world.n.08. According to the Thesaurus Linguae Latinae, which confirms the first attestation of all senses in the 1st century BCE, the sense (ii) 'humanity, philanthropy' developed from the more general sense (i) 'human nature, humanity' which refers to human nature in general. The subgraph shows that the three senses are attested at least once in passages dated 1st century BCE. However, the graph shows that the sense of 'philanthropy' dominates all other senses in the 1st century BCE. In the transition to the CE period, the sense of 'humanity' prevails regarding the number of annotations, and the two meanings coexist in the CE period. By ascending the WordNet hierarchy, we can gain deeper insight into the relationship between the two senses. The sense (ii) 'humanity, philanthropy' and the sense (i) 'human nature' are connected via two paths: sense (ii) originates from the quality.n.01 synset (i.e. 'an essential and distinguishing attribute of something or someone'); sense (i) from the attribute.n.02 synset (i.e., 'an abstraction belonging to or characteristic of an entity'). The two senses have in common the quality.n.01 synset, but the sense (ii) 'humanity, philanthropy' is directly linked to kindness.n.01 synset, and to a higher degree of the WordNet hierarchy to the morality.n.01 synset (i.e., 'concerned with the distinction between good and evil or right and wrong'). The additional information provided by including the WordNet hierarchy in the graph al-

lows us to show the type of semantic relationship between the two predominant senses of humanitas. The more general sense (i) 'human nature' specializes in its meaning in the sphere of morality, originating the sense (ii) 'philanthropy'. In the example of humanitas shown in Figure 5.2, the injected information from WordNet was exploited to analyze the semantic relationship between the meanings of the lemma humanitas. While the synset taxonomy in this example helps us track and classify phenomena of semantic change, including other types of information retrievable from the metadata can help gain further insights into the context of the semantic change. We add information about the authors' occupations in the examples shown in Figure 5.2.

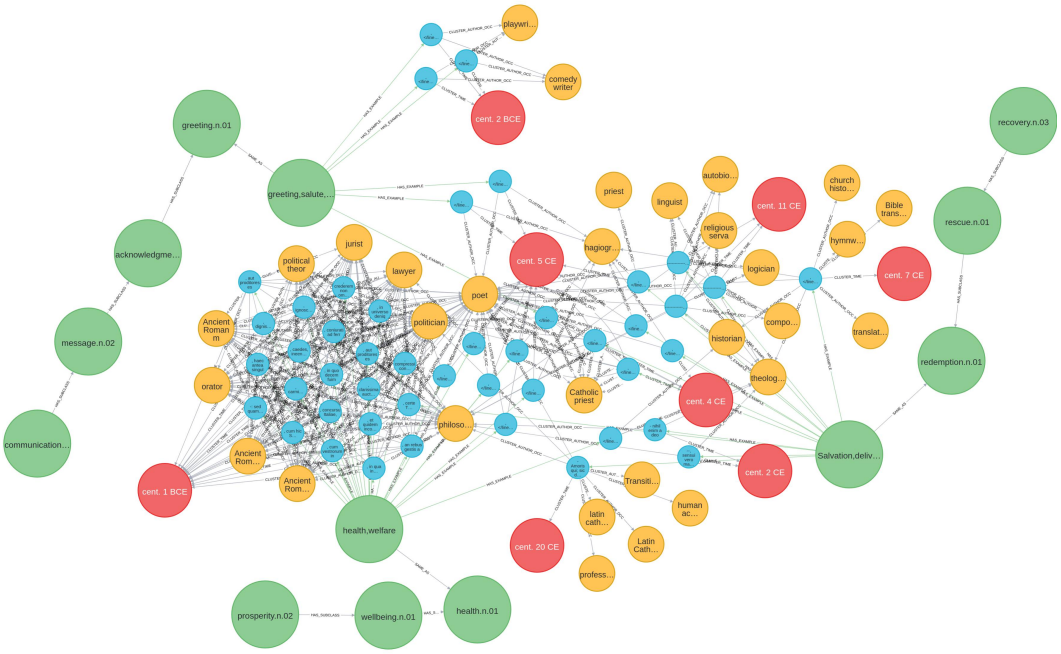
In Figure 5.2, three examples of subgraphs are shown. The three graphs refer, respectively, to the encoded information for the Latin lemmas beatus, poena, and salus. In particular, we filtered for nodes of type Text (blue nodes), Century (red nodes), Synset (green nodes), and Occupation (yellow nodes). We clustered the Text nodes by occupation and century, i.e., we created an explicit link between nodes of type Text and nodes of type TimePoint and between nodes of type Text and nodes of type Occupation. The graphs show some trends in semantic change, all related to Christianity. The lemma beatus was annotated in the SemEval dataset with five senses: (i) 'happy,' (ii) 'fortunate', (iii) 'rewarded', (iv) 'rich', and (v) 'blessed'. The graph shows that the senses (i) 'happy', (ii) 'fortunate', (iii) 'rewarded', and (iv) 'rich' all emerge starting from the 1st century BCE in the annotated dataset. On the other hand, sense (v) 'blessed' emerges later with the advent of Christianity, as we can see in correspondence with the CE nodes. In this case, there seems to be a replacement of the previous senses in favour of the Christian sense. Additionally, if we consider the nodes of type Occupation, a noticeable difference emerges between the two (groups of) meanings: in the cluster of occupation nodes connected to the Christian sense, we can observe profiles related to theological and religious activity, e.g., priests, hagiographers, which do not appear to be connected to the other senses. The same type of observations can be made for salus, which initially had the meanings (i) 'health' and (ii) 'greeting', and, subsequently, developed the Christian sense of (iii) 'salvation, deliverance from sins'. However, in this case, we can notice the difference with beatus in the type of semantic change, as the new meaning (iii) 'salvation' replaces or dominates the previously attested meanings but continues to coexist with them. The lemma *poena* also presents an example of semantic change in which the new meaning does not entirely



(a) Subgraph for beatus. The synsets for beatus are: (i) beatified.s.01: Roman Catholic; proclaimed one of the blessed and thus worthy of veneration, (ii) blessed.s.05: enjoying the bliss of heaven, (iii) rich.a.01: possessing material wealth, (iv) fortunate.a.01: having unexpected good fortune, (v) ample.s.02: affording an abundant supply, (vi) happy.a.01: enjoying or showing or marked by joy or pleasure or good fortune.



(b) Subgraph for poena. The synsets for poena are: (i) retribution.n.01: a justly deserved penalty, (ii) suffering.n.04: feelings of mental or physical pain, (iii) agony.n.01: intense feelings of suffering; acute mental or physical pain.



(c) Subgraph for salus. The synsets for salus are: (i) health.n.01: a healthy state of well-being, (ii) redemption.n.01: (Christianity) the act of delivering from sin or saving from evil, (iii) greeting.n.01: an acknowledgement or expression of goodwill.

Figure 5.2: Subgraph for beatus, poena and salus.

replace the previous ones. The new sense of "suffering, pain", which emerges in the CE nodes, continues to coexist with the sense of "punishment", which was attested since the 1st century BCE in the annotated dataset. In the case of *poena*, the contrast between the two clusters of occupation nodes is even more evident. The sense of punishment is often associated with authors classified as related to the legal world, e.g., legislator, lawyer, and jurist. In contrast, nodes related to the Christian and theological world appear in the case of salvation, e.g., theologian, priest, and presbyter. The graphs confirm well-known semantic changes prompted by the advent of Christianity, which has invested many words already in use in pre-Christian Latin with new meanings closely related to the Christian world. Moreover, the lemmas illustrate the different types of interaction between older and new senses described in literature [271]: in some cases, the two senses can continue to coexist, as for the lemmas salus and poena (a phenomenon called "layering" [272]); in others, as for the lemma beatus, the relationship between the new sense and the older ones is unbalanced as it becomes more prominent in a society invested in Christian values.

5.4 GraphBRAIN API

The GraphBRAIN technology can be appreciated thanks to the corresponding website showing its main functionalities. However, it raised the need to implement the technology into other applications in order to use GraphBRAIN as the representation layer of a more complex system. For this purpose, I developed an API that can be integrated as a jar file. GraphBRAIN makes use of a graph database to store information and a relational database to store logs associating operations (creation, update, deletion) with users. The API is independent of the specific database location since it only needs to store data according to given graph schemas, and log information. The API is parameterised according to:

- graph database
- relational database
- graph schema(s)

We provide a *default* database that is the one in which the website is hosted. In that case, the available schemas can be reused, and graph information can be accessed through the website. Through the website, information is filtered according to the schema that we decide to work on. In the context of an application, this could not be guaranteed in principle. For instance, all the insertions and updates must be checked over a graph schema that governs the possible operations. Without delving into all the details, I describe here the rationale behind the insertion operation, the most important one in order to deal with the technology.

5.4.1 Batch Insertion

Batch insertion is the operation of inserting multiple graph information without relying on any visual editor. This is the basic operation for intelligent information retrieval and management. Consistency with graph schemas must be guaranteed. For this purpose, batch insertion must always be performed with respect to a graph schema, which we assume to be noise-free. As for Cypher, we need to distinguish the creation of nodes and arcs. Specifically, it is easy to imagine that nodes must be inserted before in order to have information available to create arcs. When creating nodes, Neo4j assigns an ID

to each of the distinct nodes, and this information can be used to identify the source and object of arcs.

I designed a formalism to express nodes and arcs to be inserted into the KG. It is a JSON-like in which the following rules hold:

- nodes first, then arcs,
- all nodes must contain an "identity" field,
- properties are embedded into nodes and arcs with the "properties" field,
- arcs contain "subject" and "object", identified mapped with "identity" of nodes.

In Figure 5.3 an example is reported. Before inserting information, the consistency with respect to a graph schema must be verified. If inconsistencies are detected, the full KB is considered invalid and nothing will be inserted. Conversely, when consistency is verified, nodes are inserted and the IDs are stored. Storing the IDs of the newly added nodes allows to building of a map between them and the corresponding identities. In this way, through a replacement operation, the selection of nodes is performed by following Neo4j IDs rather than the "identity" field. This operation allows the retrieval of information by an indexed (and optimized) variable. The final queries for the toy example are shown in Figure 5.4.

5.5 Discussion and Limitations

In this chapter, I have listed some of the contexts in which GraphBRAIN technology is highly advantageous in solving problems that can be represented in some settings. All these demonstrative cases show some common advantages like flexible graph representation, which is in principle always possible, the availability of combining reasoning with graph-based inferences, and vocabulary alignment. More specifically, in the Independent Learning setting, I introduced a novel technique for learning objects' recommendations; in the Digital Libraries field, an ontology collecting and expanding standard resources; and in the linguistic field a semantic change analysis based only on data integration and graph algorithms. Although showing high flexibility and interoperability, there are still setbacks or difficulties that cannot easily be removed in the process of adopting GraphBRAIN for graph-based knowledge representation tasks.

Figure 5.3: Example of JSON representation of nodes and arcs

```
{"jtype": "node", "identity": 0, "label": "Document",
    "properties": {"name": "cent. 20 BCE", "title": "century"}}
{"jtype": "node", "identity": 1, "label": "Document",
    "properties": {"name": "", "description": "", "title": -2000}}
{"jtype": "node", "identity": 2, "label": "Month",
    "properties": {"name": "", "description": "", "month": 5}}
{"jtype": "node", "identity": 3, "label": "Year",
    "properties": {"year": "1997", "name": ""}}
{"jtype": "node", "identity": 4, "label": "Building",
    "properties": {"name": "A"}}
{"jtype": "node", "identity": 5, "label": "Town",
    "properties": {"name": "B"}}
{"jtype": "node", "identity": 6, "label": "Person",
    "properties": {"name": "A"}}
{"jtype": "node", "identity": 7, "label": "User",
    "properties": {"name": "A"}}
{"jtype": "relationship", "subject": 0, "object": 1,
"name": "aliasOf", "properties": {}}
{"jtype": "relationship", "subject": 2, "object": 3,
"name": "belongsTo", "properties": {}}
{"jtype": "relationship", "subject": 6, "object": 6,
"name": "wasInChildren", "properties": {}}
{"jtype": "relationship", "subject": 6, "object": 6,
"name": "wasInGrandChildren", "properties": {}}
```

The availability of ontologies is not guaranteed in many contexts, and producing an ontology from scratch in GraphBRAIN is as much complex and time-consuming as in every other formalism. Moreover, as shown in the next chapter, although maintaining a certain mapping and alignment, not all the predicates of RDF or RDF Schema (RDFS) can be translated onto GraphBRAIN, and versa. For this reason, in contexts in which the SW-based conceptualization is available, it can be the case that the approach looks limited for these representation aspects. Moving into GraphBRAIN requires to constraint the language in order to open for new solutions.

Figure 5.4: Example of Cypher query

```
CREATE
(n0:Document {identity: '0', name:'cent. 20 BCE', title:'century'}),
(n1:Document {identity: '1', name:'', description:'', title:'-2000'}),
(n2:Month {identity: '2', month:'5', name:'', description:''}),
(n3:Year {identity: '3', year:'1997', name:''}),
(n4:Building {identity: '4', name:'A'}),
(n5:Town {identity: '5', name:'B'}),
(n6:Person {identity: '6', name:'A'}),
(n7:User {identity: '7', name:'A'})
return ID(n0), ID(n1), ID(n2), ID(n3), ID(n4), ID(n5), ID(n6), ID(n7)

MATCH (n0) WHERE ID(n0)=102 MATCH (m0) WHERE ID(m0)=468
    CREATE (n0)-[:aliasOf]->(m0) WITH n0, m0

MATCH (n1) WHERE ID(n1)=512 MATCH (m1) WHERE ID(m1)=513
    CREATE (n1)-[:belongsTo]->(m1) WITH n1, m1

MATCH (n2) WHERE ID(n2)=516 MATCH (m2) WHERE ID(m2)=516
    CREATE (n2)-[:wasInChildren]->(m2) WITH n2, m2

MATCH (n3) WHERE ID(n3)=516 MATCH (m3) WHERE ID(m3)=516
    CREATE (n3)-[:wasInGrandChildren]->(m3)
```

Chapter 6

SW Mapping

The SW and its data representation, RDF, were conceived for different purposes than the traditional uses of DBs. The latter focuses on efficiency and scalability, while the SW focuses on interoperability and data availability. The integration between these two technologies, designed to take advantage of the strengths of both, has attracted interest since the early days of the WWW because, while the SW provides a means to share and integrate data, various factors often prevent the public distribution of data in many contexts. However, problems are posed by the structural differences between the graph models used in the SW and GB ones. Here I propose a methodology for mapping GB data into SW-compliant ones. Given the inherent limitations of fully automatic solutions, I propose a semi-automatic solution in which the mapping follows some generic rules but is personalised according to the user's previous knowledge.

6.1 GB/RDF Mapping

Since the SW setting requires every element in the KB to be uniquely identified by a URI, we defined the prefix *http://graphbrain.it/ontologyName* (abbreviated as *gb*) and obtained the URI of a resource as the concatenation of such a prefix with its element identifier in GraphBRAIN. Then, a set of translation rules has been defined to map GBs ontologies onto OWL ones.

6.1.1 Schema Rules

The entities, attributes, and relationships in a GBS are mapped onto classes, data properties, and object properties (respectively) in an OWL ontology using the obvious corresponding OWL elements *owl:class*, *owl:objectProperty* and *owl:dataTypeProperty*. While each object property must have a single domain and range, GBS allows several subject-object pairs for each relationship. To handle this, for each subject-object pair available for a relationship *rel*, we introduce a different *owl:objectProperty* named *rel_subject_object*, assigning its subject as its *owl:domain* and object as its *owl:range*. We then state that all of these newly added object properties are specializations of the *owl:objectProperty* named *rel*, with the domain given by the union of the OWL classes identified by the subjects and the range given by the union of the OWL classes identified by the objects.

Suppose the GBS relationship *rel* can be established between two subject-object pairs: (ClassA,ClassB) and (ClassC,ClassD), it translates to

ObjectProperty: rel
  Domain: ClassA or ClassC
  Range: ClassB or ClassD

ObjectProperty: rel_ClassA_ClassB
  Domain: ClassA
  Range: ClassB

ObjectProperty: rel_ClassC_ClassD
  Domain: ClassC
  Range: ClassD

where *rel_ClassA_ClassB* and *rel_ClassC_ClassD* are specializations of *rel*. You can see an example of this strategy in Fig. 6.1. Note that *rel* alone in OWL would allow the establishment of the relationship between an individual of ClassA and one of ClassD in the KB, which would not be allowed by the original GBS relationship. Adding the other two relationships makes the OWL KB inconsistent.

For relationship attributes, we use an implicit reification: we introduce a class for

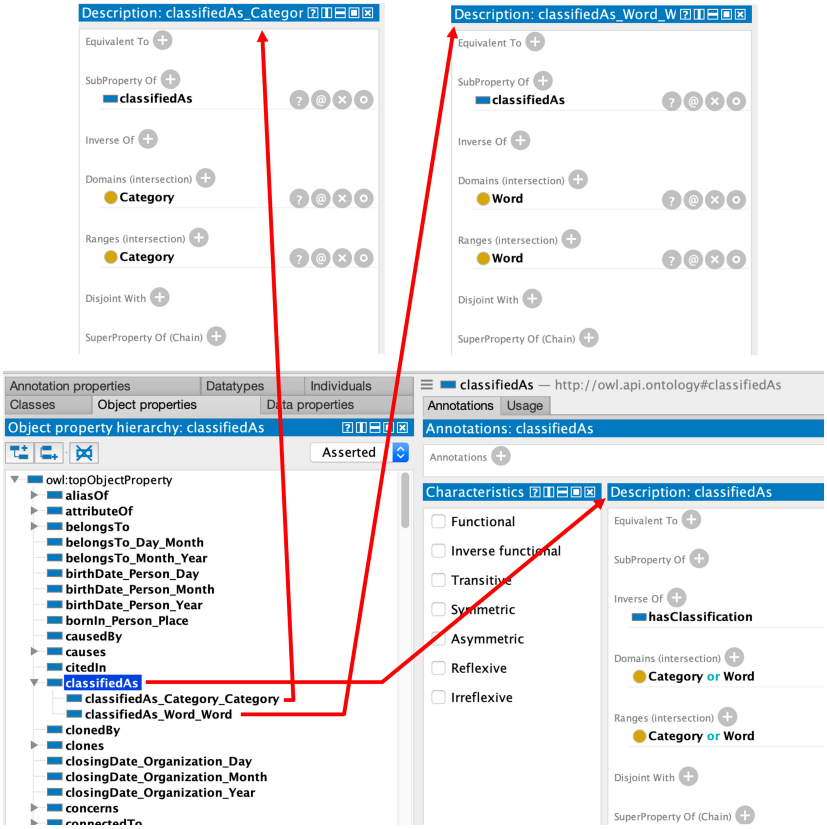


Figure 6.1: Protégé view of the classifiedAs object property and its specializations.

each relationship in the schema and the Relationship class, which generalizes all of them. This allows us to represent the attributes of relationships as datatype properties of these newly reified classes. This is compliant with the official W3C guidelines [52]. Then, for each occurrence of the relationship, an individual representing the instance of the relation is created for the corresponding class with links toward the subject and object of the relation (*owl:subject* and *owl:object* respectively) and other links to all items representing the original relationship attributes.

6.1.2 Data Mapping

After discussing how to map ontologies from GBS to OWL, we now describe how to map instances in the LPG graph to the RDF. One of the main issues in merging external resources with local DBs is that one cannot know that their ontology definition of a concept corresponds to another available on the Web. More specifically, to the best of our knowledge, there is no solution in the literature for automatically aligning such vast and (possibly) different ontologies. So, I propose a semiautomatic mapping in which the user must specify how GBS concepts are to be mapped onto those available

in the SW. I propose the use of RML by differentiating the type of values of some parameters. RML is used to map various languages for relational DBs but not graph DB. For this purpose, we use the *rml* : *logicalSource* directive by specifying a Cypher query to be applied to all nodes of a certain type as rml:referenceFormulation formally (using turtle format):

rml:logicalSource [rml:referenceFormulation ql:CypherPattern;rml:iterator "label name in GraphDB"];

Next, we apply R2RML directives (rr:subjectMap and rr:predicateObjectMap) to formalize the mapping with subject, predicate, and object of the ontology with which we want to map the DB graph entities found by the query execution. Let us introduce a small example that is useful for understanding how each single element is mapped. Consider an LPG-based KG about movies (node label *:Movie*) and people (node label *:Person*), where movies have a title (property "title") and a release date (property "release"), and persons have a name (property "name") and a birth date (property "born"). Both kinds of nodes have a numeric code (property "id") to uniquely identify their instances. There is a relationship between people and the movies they have acted in (type *:actedIn*).

```
(:Person{id:1, name:"Al Pacino", born:date("1940-04-25")})
(:Person{id:2, name:"Robert De Niro", born:date("1943-08-17")})

(:Movie{id:1, title:"The Godfather Part II", release:date
   ("1975-06-20")})

(:Person{id:1})-[:actedIn]->(:Movie{id:1})
(:Person{id:2})-[:actedIn]->(:Movie{id:1})
```

Listing 6.1: Example of simple graph data.

```
@prefix person: <http://loc.example.com/person/>.
@prefix movie: <http://loc.example.com/movie/>.

person:1 schema:name "Al Pacino"; schema:birthDate
   "1940-04-25"^^xsd:date.
person:2 schema:name "Robert De Niro"; schema:birthDate
   "1943-08-17"^^xsd:date.
```

```
6
7  movie:1
8  schema:name "The Godfather Part II";
9  schema:datePublished "1975-06-20"^^xsd:date;
10 schema:actor person:1, person:2.
```

Listing 6.2: Example of mapping based on an extension of RML.

Assuming that we want to map the LPG fragment in Listing 6.1 to the RDF using the classes and properties from schema.org (https://schema.org/, accessed on 9 September 2023) vocabulary. Listing 6.2 shows the desired result. To obtain this mapping, we must specify that LPG attribute "name" is mapped onto "schema:name", "born" onto myquoteschema:birthDate, and so on. We assume that the class and property names are unique. For the movie example, we specify the "PersonMapping", in which all elements related to the label "Person" (in the LPG graph) must be specified. In this section, we specify how to map all the attributes of nodes of type *Person* ("name" and "born", among the others). After doing the same for the other class in the example (*Movie*), we may specify how to map the relationship "actedIn". We understand that it has the same meaning as the standard object property *schema:actor*. In the mapping, we also specify the mappings for the subject and the object. The full code of the mapping is shown in Listing 6.3. Let us briefly analyze it line by line. R2RML and RML mappings consist of a set of triple maps, each representing a source of triples for the target RDF graph. Listing 6.3 defines 3 triple maps: <#PersonMapping>(lines 8–20), <#MovieMapping> (lines 22–33), and <#ActedInMapping> (lines 35–46). As per the RML specification, each triple map is based on a different logical source which, in our case, is a Cypher pattern, i.e., an expression that can be used in the MATCH clause of a Cypher query. <#PersonMapping>maps any LPG node with the label:Person to an IRI of the type schema:Person built from the property id with a template mechanism. Its LPG properties name and born are mapped to the RDF properties schema:name and schema:birthDate, respectively. Similarly, <#MovieMapping>maps LPG nodes with the label:Movie onto IRIs of the type schema:Movie and their LPG properties onto the corresponding RDF properties. Finally, <#ActedInMapping>maps LPG arcs labeled :actedInto triples connecting the two IRIs corresponding to the movie and the actor with the property schema:actor(note that the RDF property has the opposite direction

with respect to the :actedIn LPG relationship).

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rml: <http://semweb.mmlab.be/ns/rml#>.
@prefix ql: <http://semweb.mmlab.be/ns/ql#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix schema: <http://schema.org/>.
@base <http://example.com/ns#>.

<#PersonMapping> a rr:TriplesMap;
rml:logicalSource [rml:referenceFormulation ql:CypherPattern;
   rml:iterator "(p:Person)"];

rr:subjectMap
[rr:template "http://loc.example.com/person/{p.id}"; rr:class
   schema:Person];

rr:predicateObjectMap [
rr:predicate schema:name;
rr:objectMap [rml:reference "p.name"; rr:datatype xsd:string]
], [
rr:predicate schema:birthDate;
rr:objectMap [rml:reference "p.born"; rr:datatype xsd:date]
].

<#MovieMapping> a rr:TriplesMap;
rml:logicalSource [rml:referenceFormulation ql:CypherPattern;
   rml:iterator "(m:Movie
)"];

rr:subjectMap [rr:template "http://loc.example.com/movie/{m.id
   }"; rr:class schema:
Movie];

rr:predicateObjectMap [
rr:predicate schema:name;
```

```
rr:objectMap [rml:reference "m.title"; rr:datatype xsd:string]
], [
rr:predicate schema:datePublished;
rr:objectMap [rml:reference "m.released"; rr:datatype xsd:date]
].

<#ActedInMapping> a rr:TriplesMap;
rml:logicalSource
[rml:referenceFormulation ql:CypherPattern; rml:iterator "(p)-[a
   :actedIn]->(m)"];

rr:subjectMap [rml:reference "m"; rr:parentTriplesMap <#
   MovieMapping>];

rr:predicateObjectMap [
rr:predicate schema:actor;
rr:objectMap [rml:reference "p"; rr:parentTriplesMap <#
   PersonMapping>]
].
```

Listing 6.3: Example of RML-based mapping.

Publishing a GraphBRAIN KG in the SW by entirely exporting it would require the translation to be run again after each modification of the KG. An approach that is more suitable for dynamic KGs would be to expose the KG as an online SPARQL endpoint, from which data can be fetched at need by any SW-compliant application. Of course, the mapping technicalities between the different underlying representations should be transparent to the users. In the following text, we identify the resources by generic property values, as is done in traditional DB settings, but not necessarily by URIs. In the following text, we refer again to the movie KG and to the mapping in Listing 6.3 and to the excerpt of a possible movie dataset shown in Figure 3. Two actors, “Al Pacino” and “Robert De Niro”, both acted in “The Godfather Part II”, but “Al Pacino” also acted in “The Godfather”, whose sequel is “The Godfather Part II”. The two actors are labelled as *Person*, and the two films as *Movie*. Note that we defined an equivalent concept for the relationship “actedIn”, but not for *sequel* (see

Listing 6.3).

We focus now on three possible requests representing some of the most common queries.

- return the type of node having a property.

  This corresponds to the following SPARQL query:

  SELECT DISTINCT ?object ?objectclass WHERE ?object [property] [value]. ?object rdf:type ?objectclass

  Our parsing strategy recognizes the query type thanks to the keyword rdf:type, and then, using simple pattern matching of specific tags in the query, it extracts the property [property], and the value [value], e.g., suppose the query asks for the type of node whose name (http://schema.org/name) matches the string "The Godfather". We check in the mapping file whether "http://schema.org/name" corresponds to some property in the graph, and indeed, it has two correspondences: "p.name" in PersonMapping, and "m.title" in MovieMapping. PersonMapping has as a logical source *Person*, while MovieMapping has as a logical source *Movie*. So, we check the graph for all *Person* nodes with the value "The Godfather" as the property "name" and all "Movie" nodes with the value "The Godfather" as the property "title". To do this, when the parser recognizes that we are looking for the type of resource given the property, we prepare a general parametric Cypher query to retrieve this information from the graph. In our case, we execute this query, which is the result of the interpretation of the SPARQL query after the mapping process:

  MATCH (n:Movie title:"The Godfather") return labels(n) UNION ALL MATCH (n:Person name:"The Godfather") return labels(n)

  As expected, the result is the film named "The Godfather". We do not return the type extracted from the graph, but obtain its reverse mapping according to Listing 6.3. In the graph shown in Figure 3, the label of "The Godfather" is "Movie", but by reversing the mapping procedure, we return "http://schema.org/Movie". This allows SW-based applications to infer new knowledge about the specific instance provided by the graph DB.

- returns all links directly connected to a given node.

This corresponds to the following SPARQL query:

SELECT DISTINCT ?link WHERE ?object [property] [value] . ?object ?link ?outObject . UNION ?inObject ?link ?object . ?object [property] [value] .

As shown in the previous scenario, we used a property value to identify the starting node. Since we do not care about the direction of the object property, the nodes we are looking for can be connected by an incoming or an outgoing arc to the given node, and we must return the union of these two cases. In the former, the property is associated with the object—in the latter the object of the relationship. Our approach to handling this case separates the two pieces of the query when a UNION is encountered and then applies the same procedure as for the first scenario, but this time by considering any object property, not only the rdf:type. The output consists of pairs of object properties and the corresponding subjects (for incoming arcs) or objects (for outgoing arcs). Finally, since the SELECT DISTINCT ?link specifies that only the object property must be returned and it must not be repeated, the output will be a list of distinct relationships to which the starting node is connected, regardless of whether it plays the role of the subject or the object. Again, the properties are mapped back according to Listing 6.3. However, it may happen that the mapping document does not provide a mapping for a specific resource, i.e., the designer has not specified an external class (resp. object property) that can be considered to be equivalent to a class (resp. relationship) in the GraphBRAIN ontology. In these cases, we assign the URI of GraphBRAIN, i.e., http://graphbrain.it/name (where name is the name of the class or relationship) to these resources. So, we may return mixed resources, connecting standard ontological concepts with those defined by our schemas. For example, supposing the property is again *http://schema.org/name* and the value is *The Godfather*, Figure 3 shows that the film is reachable, not only through the *actedIn* relationship but also through *sequel*, for which no mapping has been provided. After parsing the query, again, *http://schema.org/name* is ambiguous, so the query is decomposed as the union of many smaller queries. After the mapping, the equivalent Cypher query is

MATCH (n:Movie title:“The Godfather”)-[r]-(m) return type(r) UNION ALL MATCH (n:Person name:“The Godfather”)-[r]-(m) return type(r)

It returns two relationships from the graph: "sequel" and "actedIn". The latter will be mapped, but the former will not, yielding the final result *http://graphbrain.it/sequel*, *http://schema.org/actor*.

- export a piece of the KG.

  This is the most general scenario, representing an example of how to enrich the SW knowledge by exporting data (pieces of KG) from GraphBRAIN. It is associated with the query:

  MATCH (n:[Subj])-[r:[rel]]->(m:[Obj]) RETURN labels(n), properties(n), type(r), labels(m), properties(m)

  Here, we extract all triples consisting of a node of type [Subj] that is related to a node of type [Obj] through a [rel] arc from the graph. Suppose [Subj] is "Person", [rel] is "actedIn", and [Obj] is "Movie". This involves three elements of the mapping in Listing 6.3: PersonMapping, MovieMapping, and ActedInMapping. The former indicates the class in the SW that is equivalent to "Person" and also maps two of its properties ("born" and "name"). The second determines the concept equivalent to class *Movie* and also maps its property "title". Finally, the latter specifies how to map the relationship. ActedInMapping has as a logical source (p)-[a:actedIn]→(m). p and m have their mappings as well: the former is related to MovieMapping, and the latter to PersonMapping. Hence, we know that the subject must be a *Person* in the graph, and the object must be a *Movie*. This completes the mapping of all extracted elements.

```
1  Prefix(:=<#>)
2  Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
3  Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
4  Prefix(xml:=<http://www.w3.org/XML/1998/namespace>)
5  Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
6  Prefix(gb:=<http://www.gb.it#>)
7  Prefix(schema:=<http://schema.org/>)
8  Ontology(<>
9
10 Declaration(Class(<#Movie>))
11 Declaration(Class(<#schema:Person>))
```

```
Declaration(ObjectProperty(<#schema:actor>))
Declaration(DataProperty(<#schema:birthDate>))
(DataProperty(<#schema:datePublished>))
(DataProperty(<#schema:name>))
(NamedIndividual(<gb:AlPacino>))
(NamedIndividual(<gb:RobertDeNiro>))
(NamedIndividual(<gb:TheGodfather>))
(NamedIndividual(<gb:TheGodfather2>))

# Named Individuals

ClassAssertion(<#schema:Person> <gb:AlPacino>)
ObjectPropertyAssertion(<#schema:actor> <gb:AlPacino> <gb:
   TheGodfather>)
ObjectPropertyAssertion(<#schema:actor> <gb:AlPacino> <gb:
   TheGodfather 2>)
DataPropertyAssertion(<#schema:birthDate> <gb:AlPacino>
   "25/04/1940")
DataPropertyAssertion(<#schema:name> <gb:AlPacino> "Al
   Pacino")

ClassAssertion(<#schema:Person> <gb:RobertDeNiro>)
(<#schema:actor> <gb:RobertDeNiro> <gb:TheGodfather2>)
DataPropertyAssertion(<#schema:birthDate> <gb:RobertDeNiro>
   "17/08/1943")
DataPropertyAssertion(<#schema:name> <gb:RobertDeNiro> "
   Robert De Niro")

ClassAssertion(<#schema:Movie> <gb:TheGodfather>)
DataPropertyAssertion(<#schema:datePublished> <gb:
   TheGodfather> "21/09/1972")
DataPropertyAssertion(<#schema:name> <gb:TheGodfather> "The
   Godfather")

ClassAssertion(<#schema:Movie> <gb:TheGodfather2>)
```

```
39 DataPropertyAssertion(<#schema:datePublished> <gb:
       TheGodfather2> "20/06/1975")
40 DataPropertyAssertion(<#schema:name> <gb:TheGodfather2> "The
       Godfather 2")
41 )
```

Listing 6.4: Example of a mapped KB from a triple.

The OWL/RDF KB is created with the OWL API [273]. The results of the mapping and export are given in Listing 6.4.

6.2 Mapping Considerations

Some aspects of the mapping need to be considered carefully. Before moving into the SW perspective, the Closed World Assumption (CWA) guaranteed many positive properties like consistency, name disambiguation, and soundness, and reasoning does not lead to unexpected results. These aspects cannot be guaranteed any longer when *opening* the world. Indeed, as soon as an instance is enriched with *semantics*, the information originally available in GraphBRAIN is only a part of the whole knowledge. The membership to an existing concept may lead to different inferences, making the process of reasoning much more unpredictable. The API mentioned in 5.4 actually *closes* the world. The API is intended to facilitate operations within the context of a schema-aware graph database and for this sake, I consider it more beneficial to not introduce unexpected inferences while looking for inferences.

6.3 Evaluation

In this mapping strategy, it is not quite possible to define quantitative evaluation or computational details, but an expressiveness comparison can be done, following some of the considerations already mentioned in Section 3.2.2. Malinverni et al. [274], although referring to a specific use case, modelled a mapping strategy from GIS data to RDF. Although the connection, the mapped data have been specifically assigned URIs that are unique and hence not connected to anything. This mapping does not change (or add) semantics to the initial data but allows reasoning in the CWA perspective.

Conversely, in the context of DB mapping, there has been much effort in automatically mapping logical models in RDF but the lack of personalization and the capability of linking instances gave the opportunity to build the proposed mapping.

Chapter 7

Schema Design & Evaluation

Although yielding competitive performance and interpretability, as previously pointed out, graph databases lack a schema for sharing the vocabulary and preserving consistency in the data. For this scope, we enrich the graph database with a manually constructed ontology defining labels and the relationships among them. However, this manual process is error-prone, time-consuming and may lead to ambiguity or incompleteness. In this chapter, I introduce a data-driven approach based on simulation to assist ontology designers in refining schemas for graph-based structures, specifically within the context of modal graph schemas [275]. I validated the approach using the GraphBRAIN datasets, supported by manually curated schemas for evaluation, as well as two standard datasets: Twitter Neo4j and the Lehigh University Benchmark. The method aims to enhance the accuracy and efficiency of ontology design, while also speeding up the design time.

7.1 Graph Databases and Schemas

I am going to redefine the very basic graph definitions as well as the notational body of graph schemas presented by Mennicke [275]. While there are subtle differences in the manifold of graph database models, they all share a basic yet directed and labeled graph structure [40]. Let Σ be a set of labels, used as arc inscriptions, being properties and/or relations. A *(directed edge-labeled) graph* $\mathcal{G}$ is a triple $(\mathcal{V},\Sigma,\rightarrow)$ where $\mathcal{V}$ is a finite set of nodes and $\rightarrow\subseteq \mathcal{V} \times\Sigma\times \mathcal{V}$ a (necessarily) finite labeled edge relation. I denote an a-labeled edge from node v to node w as $v \xrightarrow{a} w$ (formally, $(v, a, w) \in\rightarrow$). Nodes represent

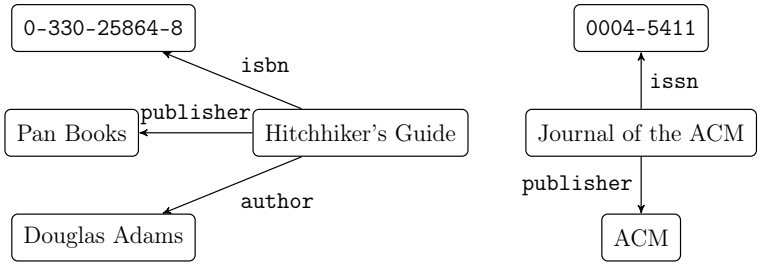


Figure 7.1: An Example of Graph Database: A Tiny Excerpt of a Bibliographical Database

entities and literals while edges provide relationships/properties between them. The graph depicted in 7.1 shows an excerpt of a bibliographical database, having a book (i.e., Hitchhiker's Guide) and a journal (i.e., Journal of the ACM) and information about them.

For the rest of this chapter, assume graph DB $= (\mathcal{V}, \Sigma, \rightarrow)$ to be our *graph database*. At first glance, any graph schema for DB can be represented by a graph. After all, I want to mention classes in DB and their relationships to other classes, which calls for classes to be nodes and (labeled) edges between them. Let, therefore, assume a first graph schema model as graphs $\mathcal{S} = (\mathcal{C}, \Sigma, \longrightarrow_s)$ in which I call the elements of $\mathcal{C}$ the *classes* of $\mathcal{S}$. The edge relation of $\mathcal{S}$ provides information about the *allowed* properties of the entities belonging to the classes. Conversely, if there is no edge $c_1 \xrightarrow{r}_s c_2$, then entities e_i $(i = 1, 2)$ belonging to class c_i *must not* be connected by relation r. Given DB and $\mathcal{S}$, I relate the entities and (potentially) literals (i.e., the nodes) of DB to the classes mentioned in $\mathcal{S}$ by means of relations $\mathcal{R} \subseteq \mathcal{V} \times \mathcal{C}$. By design, a single data node may belong to more than one class. Not every such relation should be considered since I want to exclude the classification of DB entities participating in relationships that are not allowed by the respective classes in $\mathcal{S}$.

Graph Schema Conformance [275]

For graph database DB $= (\mathcal{V}, \Sigma, \rightarrow)$ and graph schema $\mathcal{S} = (\mathcal{C}, \Sigma, \underset{s}{\rightarrow})$, a relation $\mathcal{R} \subseteq \mathcal{V} \times \mathcal{C}$ is a *conformance relation for DB and* $\mathcal{S}$ if for all $(v, c) \in \mathcal{R}$ and $a \in \Sigma$:

- for every $w \in \mathcal{V}$ with $v \xrightarrow{a} w$, there is a $d \in \mathcal{C}$ such that $c \underset{s}{\xrightarrow{a}} d$ and $(w, d) \in \mathcal{R}$ and
- for every $u \in \mathcal{V}$ with $u \xrightarrow{a} v$, there is a $b \in \mathcal{C}$ such that $b \underset{s}{\xrightarrow{a}} c$ and $(u, b) \in \mathcal{R}$.

A data node $v \in \mathcal{V}$ *conforms to* class $c \in \mathcal{C}$, written as $v \Vdash^S_{DB} c$, if there is a conformance relation $\mathcal{S}$ such that $(v, c) \in \mathcal{R}$. For class $c \in \mathcal{C}$, I denote the set of all data nodes

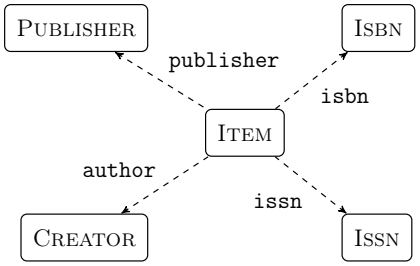


Figure 7.2: An Initial Graph Schema for Bibliographical Data

captured by class c by $\Vdash^S_{DB} c := \{v \in \mathcal{V} |\ v \Vdash^S_{DB} c\}$

Any conformance relation, including $\Vdash^S_{DB}$, is a *dual simulation* [275, 276]. Dual simulations have been considered as a feasible structure-preserving alternative to the usual graph matchings based on homomorphisms or even (subgraph) isomorphisms [277]. Most prominently, the algorithmics of (dual) simulations present polynomial time procedures [278, 279].

Graph schemas together with the conformance relation (7.1) provide already two modalities for relationships between classes and their entities. First, an edge present in the graph schema refers to an *allowed* relationship in the data while an edge that is not present in the schema is a *forbidden* one. Consider the graph schema for bibliographical data in 7.2. The entities of 7.1 conform to the schema by the conformance relation

$\mathcal{S} = \{(\text{Hitchhiker's Guide}, \textsc{Item}), (\text{Douglas Adams}, \textsc{Creator}), \dots\}$.

When thinking about classes and their properties, I make another distinction by requiring the presence of certain relationships as *mandatory*. Having *mandatory* as an additional modality yields the final graph schema model, namely *modal graph schemas* [275], which I subsequently used as the basis for graph schema creation as well as refinement.

(Modal) Graph Schema [275] Let $\text{DB} = (\mathcal{V}, \Sigma, \rightarrow)$ be a graph database. A *(modal) graph schema* is a quadruple $\mathcal{S} = (\mathcal{C}, \Sigma, \rightarrow^{\Diamond}_s, \rightarrow^{\Box}_s)$ such that $\mathit{skeleton}(\mathcal{S}) = (\mathcal{C}, \Sigma, \rightarrow^{\Diamond}_s)$ is a graph and $\rightarrow^{\Box}_s \subseteq \rightarrow^{\Diamond}_s$.

Call $\mathcal{R} \subseteq \mathcal{V} \times \mathcal{C}$ a *(modal) conformance relation for DB and* $\mathcal{S}$ if (i) it is a conformance relation for DB and $\mathit{skeleton}(\mathcal{S})$ (according to 7.1) and (ii) for all $(v, c) \in \mathcal{R}$ and $a \in \Sigma$:

- for every $d \in \mathcal{C}$ with $c \xrightarrow{a}{}^{\Box}_s d$, there is a $w \in \mathcal{V}$ such that $v \xrightarrow{a} w$ and $(w, d) \in \mathcal{R}$ and

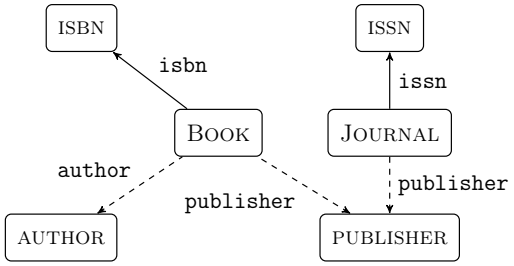


Figure 7.3: A Modal Graph Schema Distinguishing Books and Journals

- for every $b \in \mathcal{C}$ with $b \overset{a}{\rightarrow}_s^{\Box} c$, there is a $u \in \mathcal{V}$ such that $u \overset{a}{\rightarrow} v$ and $(u, b) \in \mathcal{R}$.

$\Vdash_{DB}^{S}$ is the largest modal conformance relation and, thereby, $v \Vdash_{DB}^{S} c$ means v conforms to class c.

I have overloaded the symbol $\Vdash_{DB}^{S}$ since the original (non-modal) conformance relation is a special case. The $\Box$-modality marks the so-called *must-edges* and is represented as solid edges (cf. 7.3), while the $\Diamond$-modality refers to the *may-edges* of $\mathcal{S}$ that are represented by dashed edges. By analogy, a modal graph schema $\mathcal{S}$ covers the three desired modalities, namely *forbidden* (absent edges), *allowed* (may-edges), and *mandatory* (must-edges) relationships. Note, since every must edge is a may edge, every mandatory relationship is also an allowed one.

Classes BOOK and JOURNAL, represented in 7.3, are refinements of ITEM in 7.1. Since for BOOK, property `isbn` is required, only data node Hitchhiker's Guide (in 7.1) will be captured by class BOOK while Journal of the ACM cannot be regarded as BOOK anymore. Note, implicitly entities of class JOURNAL must not have `author`-properties in this example.

Further note, the conditions in 7.1 (ii) are quasi-symmetric (on $\rightarrow_s^{\Box}$) to the requirements of 7.1. These additional requirements provide the necessary meaning of the $\Box$-modality, namely that edges required for a class are covered by an associated graph database entity.

As a final remark, the definitions of graph schema conformance and modal schema conformance coincide in the special case of modal graph schemas S such that $\rightarrow_s^{\Box} = \emptyset$. If, on the other hand, $\rightarrow_s^{\Diamond} = \rightarrow_s^{\Box}$, then a modal conformance relation is called a *dual bisimulation*. Note that even with the additional constraints, the algorithmics of (modal) graph schema conformance remains tractable [280–282]. It is the tractability of the conformance relation I exploit throughout the graph schema generation process,

as introduced throughout the next section.

7.2 Graph Schema Creation and Refinement

With rare exceptions [222], graph databases in the wild do not feature full-fledged schema information, let away graph schemas according to 7.1. However, the methodological perspective of a (modal) graph schema conformance allows for describing a step-wise refinement process towards a meaningful graph schema. As early as in [275], the *unit graph schema* $\mathcal{U}$ was mentioned to be a valid universal starting point for schema creation. $\mathcal{U}$ consists of a single class u and as many self-loops as there are elements in the labeling alphabet Σ. Of course, all self-loops are considered may-edges. Since all graph databases conform to the unit schema, it serves as a theoretical basis for a step-wise refinement process splitting up classes until a meaningful schema has been achieved. Note, every data node $v \in \mathcal{V}$ belongs to class u.

Let $\mathcal{U} = (\{u\}, \Sigma, \{u\} \times \Sigma \times \{u\}, \emptyset)$ be the *unit graph schema*. Then for all graph databases DB$= (\mathcal{V}, \Sigma, \mathcal{E})$, $\mathcal{R} = \{(v, u) \mid v \in \mathcal{V}\}$ is the maximal modal conformance relation for DB and $\mathcal{U}$. The unit class u collects all data nodes, just like `owl:Thing` in OWL ontologies [232]. Of course, taking the unit schema as the universal starting point is unsatisfactory when graph data already exists. For a sensible graph schema, a lot of class splitting may be anticipated. Therefore, let us describe a more viable yet data-driven option for graph schema initialization in 7.2.1. I exhibit the core property of (modal) graph schema conformance, namely its formal basis in dual simulation which is, after all, a preorder. Later on in 7.2.2, I define graph schema refinement operations guiding the graph schema creation process.

7.2.1 Graph Schema Initialization

The whole idea of graph schemas and their conformance relation is that classes of entities exhibit similar structure. However, if data nodes exhibit similar structures, we may be able to observe the similarities on the graph database directly. This is the cornerstone of what I refer to as *graph schema initialization*. Therefore note that dual simulation (i.e., graph schema conformance according to 7.1) is a preorder. We can use

dual simulation equivalence directly on a graph database DB to obtain the respective quotient graph schema.

Initialization by Dual Simulation Quotienting For graph database $\mathrm{DB} = (\mathcal{V}, \Sigma, \mathcal{E})$, the largest dual simulation $\mathcal{R} \subseteq \mathcal{V} \times \mathcal{V}$ on DB exists and is unique. For $v \in \mathcal{V}$, let $[v]_{/R} := \{u \in \mathcal{V} \mid (u,v), (v,u) \in \mathcal{R}\}$ and define

$$\mathcal{Q} := (\{[v]_{/R} \mid v \in \mathcal{V}\}, \Sigma, \rightarrow^{\Diamond}, \emptyset)$$

where $[v]_{/R} \xrightarrow{w}{}^{\Diamond}_{a} /R$ if $v \rightarrow_a w$. Then $\mathcal{R} := \{(v, [w]_{/R}) \mid (v,w) \in \mathcal{R}\}$ is the largest modal conformance relation and captures all data nodes $v \in \mathcal{V}$.

proof The uniqueness of the largest dual simulation $\mathcal{R}$ follows from the fact that (dual) simulations are closed under unions (cf. [275, 277]). $\mathcal{R}$ is a dual simulation since all its pairs stem from the original dual simulation R (and its symmetric closure). We need to show that (1) $\mathcal{R}$ is a model conformance relation, (2) $\mathcal{R}$ is the largest one, and (3) $\mathcal{R}$ captures all data nodes $v \in \mathcal{V}$.

Regarding (1), let $(s,t) \in \mathcal{R}$, i.e. $s \in \mathcal{V}$ and $t = [v]_{/R}$ and $(s,v) \in \mathcal{R}$ for some $v \in \mathcal{V}$. If $s \rightarrow_a s'$, then there is a node $w \in \mathcal{V}$ such that $v \rightarrow_a w$ and $(s', w) \in \mathcal{R}$ (since $\mathcal{R}$ is a dual simulation). Thus, $[v]_{/R} \xrightarrow{w}{}^{\Diamond}_{a} /R$ (by definition of $\rightarrow^{\Diamond}$). Ultimately, $(s', [w]_{/R}) \in \mathcal{R}$ since $(s', w) \in \mathcal{R}$. Since this argument holds for arbitrary pairs $(s,t) \in \mathcal{R}$ and labels $a \in \Sigma$, $\mathcal{R}$ is a modal conformance relation (also because $\rightarrow^{\Box}$ of $\mathcal{Q}$ is empty). Towards (3), note that every data node $v \in \mathcal{V}$ is captured in $\mathcal{Q}$. As dual simulation/(modal) conformance is reflexive (since it is a preorder), at least $(v, [v]_{/R}) \in \mathcal{R}$ for every data node $v \in \mathcal{V}$.

It remains to be shown that (2) $\mathcal{R}$ is the largest modal conformance relation. Let $\mathcal{R}$' be an arbitrary conformance relation and let $(s,t) \in \mathcal{R}$'. I show $(s,t) \in \mathcal{R}$ and, thereby, show that $\mathcal{R}' \subseteq \mathcal{R}$, making $\mathcal{R}$' the largest conformance relation. If $(s,t) \notin \mathcal{R}$, then $t = [v]_{/R}$ and $(s,v) \notin \mathcal{R}$. But since $\mathcal{R}$ is the largest dual simulation on DB, it contains all pairs of nodes (v_1, v_2), such that v_2 dual simulates v_1. That means, if v dual simulates s, then $(s,v) \in \mathcal{R}$. Hence, $\mathcal{R}$' can only be a modal conformance relation if all its pairs are contained in $\mathcal{R}$.

Observe that the quotient graph schema $\mathcal{Q}$ combines similar data nodes to classes. Summarizing, every graph database DB exhibits a canonical quotient graph schema, to which it conforms. The necessary (dual simulation) relation $\mathcal{R}$ also suggests initial

hierarchies between the data nodes, but solely on a structural basis. Next, a domain expert needs to review the initial schema and refine it according to the domain requirements. I provide appropriate refinement operations for the domain expert throughout the rest of this section.

7.2.2 Graph Schema Refinement

When it comes to graph schema refinement, I start with a graph database DB = $(\mathcal{V}, \Sigma, \mathcal{E})$ and an initial graph schema $\mathcal{S} = (\mathcal{C}, \Sigma, \rightarrow_s^{\Diamond}, \rightarrow_s^{\Box})$ describing the data already in a certain depth by means of the conformance relation $\Vdash_{DB}^{\mathcal{S}}$ capturing all the nodes of DB. This is a necessary assumption since I want to describe all the data by the schema at hand. Also, it is a valid assumption for the unit schema U and the quotient schema $\mathcal{Q}$ (cf. 7.2.1). But $\mathcal{S}$ can be any of the two or even a refinement thereof. I inspect classes $c \in \mathcal{C}$ through $\Vdash_{DB}^{\mathcal{S}}$. Therefore, let $\Vdash_{DB}^{\mathcal{S}} c$ be the set of nodes $\{v \in V \mid v \Vdash_{DB}^{\mathcal{S}} c\}$ classified by c. I say a data node $v \in V$ is *captured by* class $c \in \mathcal{C}$ if $v \in \Vdash_{DB}^{\mathcal{S}} c$. The subsequent refinement operations aim at producing a schema $\mathcal{S}'$ from $\mathcal{S}$ such that all nodes of DB are still captured (i.e., $\bigcup_{c \in \mathcal{C}} \Vdash_{DB}^{\mathcal{S}'} c = V$).

Merging Classes. After creating the quotient schema, there are some classes a domain expert may consider the same. Dual simulations (i.e., graph schema conformance) are quite picky. For instance, if for some books the year of publishing is reflected in the data and for some books not, then the quotient graph contains two different classes. Such classes may easily be merged by identifying the classes that may be considered too fine-grained. Let $c_1, c_2 \in \mathcal{C}$ be two classes. Merging c_1 and c_2 means that, in the new schema $\mathcal{S}'$, I identify both nodes with a single fresh class $c_{1,2}$ and remove the original classes. Class $c_{1,2}$ inherits all the may edges of c_1 and c_2. Must edges of c_1 or c_2 are only inherited if they are must edges in both classes. Note, since every must edge is a may edge by definition, even if a must edge is not transferred to $c_{1,2}$, it still persists as a may edge. Consider again the graph schema in 7.3. The schema in 7.2 merges classes Book and Journal. While properties `isbn` and `issn` are preserved, they are not mandatory anymore.

The reason why conformance and coverage of all nodes is preserved is that, intuitively speaking, the new class $c_{1,2}$ simulates both classes, c_1 and c_2. Thus, every node

that was classified as c_1 or c_2 is now captured by $c_{1,2}$.

Creating Hierarchy. Domain experts may further observe certain classes containing entities of more than one class and this single class may be considered too coarse. Then a first step towards a better schema might be to incorporate hierarchy. Suppose, class $c \in \mathcal{C}$ captures a node u we would like to distinguish from the other nodes captured by c in a subclass relationship. Therefore, let $v \in \Vdash^{\mathcal{S}}_{DB} c$ that is not equivalent to u (i.e., $(u, v) \notin R$ or $(v, u) \notin R$). If no data node (captured by c) structurally differs from u, there is no chance of distinguishing u on the schema level. The class nodes are (structurally) too similar to distinguish them any further. In such a situation, a domain expert may request one or more distinguishing properties for the data nodes.

If the two nodes (u and v) are distinct with respect to the relationships they participate in, I create a copy of c_u of class c, initially inheriting all of the adjacency of c. Let $b \in \Sigma$ be a relation such that $u \to_b u'$ (or $u' \to_b u$, resp.), but $v \to_b v'$ (or $v' \to_b v$, resp.) for no $v' \in V$. Then the domain expert has the option to make b mandatory for c_u. This has the effect that node v is not captured by the fresh class c_u while it is still a member of the original class c. Only nodes like u are now captured by c_u. Alternatively, we may also observe that there is some $a \in \Sigma$ such that $v \to_a v'$ (or $v' \to_a v$, resp.) but u does not participate in any a-edge. In that case, we may remove the a-edge from class c_u, having the same effect: since a-edges are forbidden in c_u, the class does not capture node v anymore. Everything else is not altered. Reconsidering the example schemas, class Book (7.3) is a refined version of class Item (7.2) by making `isbn` mandatory and removing the `issn`-property. Conversely, Journal has been obtained by refining Item keeping only properties `issn`, which is also made mandatory, and `publisher`.

Separating Classes. Ultimately, the two data nodes u and v may be separated into two distinct classes c_u and c_v by repeating the technique for creating a hierarchy for both nodes. This yields two distinguishable classes, just like Book and Journal. Up to a certain point, it is required to keep the original class c intact. If, alongside the schema refinement process, sufficiently many classes have been spawned so that all the data nodes originally captured by c are then captured by the fresh classes, a domain expert may decide to remove the upper class c.

Let's mention that making certain edges mandatory should be considered with care. For properties with literal values as target there is nothing to worry about. But if we made only a single `publisher`-edge mandatory, then class PUBLISHER certainly gets a different meaning and, recursively, also books and journals. Only book and/or journal publishers are then captured that also host the other type of publication. Consequently, books published by publishers with no journals would not be captured anymore. In such cases, separating class PUBLISHER into BOOK PUBLISHER and JOURNAL PUBLISHER may result in a more meaningful refinement. Throughout the next section, I discuss a case study in graph schema creation via GraphBRAIN. There, we see how the different refinement operations play along with the actual requirements given by the respective domain.

7.3 A Case Study

I applied graph schema initialization and refinements to GraphBRAIN, a dataset that already possesses a schema, providing instance information as well as *allowed* and *mandatory* properties. I used the existing schema to simulate the domain expert who is then guided through the graph schema creation process. All the experiments with simulation have been conducted with a revisit of the algorithm in [282], executed on an Intel(R) Core(TM) i7-1065G7 CPU single-core, 16GB of RAM processor. I applied the schema creation method to GraphBRAIN, being a dataset with a manually curated schema.

7.3.1 Dataset Preparation

Given the amount of data, I executed the experiments by selecting several relevant pieces. I took the more relevant nodes (ranked with Page Rank), and I applied a spreading activation function starting from the central nodes. The LPG was then mapped onto a triple-based graph by mapping properties into arcs. Without losing generality, I neglected relationship properties since they do not take part in the simulation process. Properties of nodes must belong to a class; hence, I introduced *String* to collect all property values available.

Throughout the use case, I will refer to two different excerpts from GraphBRAIN,

Table 7.1: Two excerpts from GraphBRAIN

GraphBRAIN	# node labels	# arcs	# cls	# pure cls	# non-hierarchical cls
(i)	15	25	54	40	5
(ii)	50	36	78	53	18

which I will call GraphBRAIN (i) and GraphBRAIN (ii). GraphBRAIN (i) is made up of ∼90k triples from the **retrocomputing** domain, collecting resources about computing organizations and scientific papers, but also relevant known places of scientific events. GraphBRAIN (ii) by ∼18k triples from the **lam** domain having much information on books and documents, but also on authors and other organizations. I describe schema creation in these examples.

7.3.2 Schema Initialization

The simulation-driven approach required a preliminary step of data labelling, and the initialization was provided by the simulation function, creating clusters (cls) of nodes. Without previous assumptions, each cluster may refer to a different class to be represented in the schema, and from them the final schema will emerge through the refinement steps *class creation*, *splitting*, *merging* and *hierarchy creation*. Among the clusters, I could distinguish those that straightforwardly need to be represented in the schema as they are, what I refer to as *pure clusters*. They are clusters in which only one label (i.e., a class name of the provided ontology) appears viz. all nodes in the cluster have been equally labelled. I report in 7.1 some info about the initialization of the two GraphBRAIN datasets.

The simulation function (i.e., the quotient schema) recognised 54 clusters for the first dataset and 78 for the second. Pure clusters are those that necessarily *must* be represented in the schema. Naturally, some of the pure clusters represented the same (repeated) label but I will discuss this during refinement. What deserved further attention were the impure clusters, which represented a minority in both cases. Resolving the impure clusters is infeasible to do automatically. Hence, the need for the above-mentioned refinement operators as well as the domain expert rises. The operation of recognising non-hierarchical clusters is manual, too. The numbers provided in 7.1 provide a trustworthy estimation of the effort of the schema designer for the manual curation of the schema.

From GraphBRAIN (i), I report some examples of clusters that work well as a basis for graph schema refinement:

- $[Book, Magazine, Booklet, Document]$ because $Book$, $Magazine$, and $Booklet$ are subsumed by $Document$ in GraphBRAIN;
- $[Month, Day]$ because there is a GraphBRAIN class for $TemporalSpecification$;
- $[Device, DeviceItem]$ because $DeviceItem$ is a specialization of class $Device$ in GraphBRAIN.

I also mention some non-hierarchical ones:

- $[Magazine, Book, Month]$ since $Month$ must not be grouped with *Book-like* classes
- $[Person, Organization]$ since they represent completely distinct concepts.

7.3.3 Schema Refinement

Refinement is a domain-specific and subjective task; however, with the data-driven approach, there are some operations that can naturally be performed without introducing schema concerns. As previously mentioned, not all the *pure clusters* are distinct; this happens because there are data nodes belonging to *pure clusters* having different properties from other equally-labelled nodes put into another *pure cluster*. In this case, the refinement operator is always *class merging* and properties are merged alongside the classes. It is an efficient, labour-saving and safe operation. In GraphBRAIN (ii), more than half (80%) of the *pure clusters* are in fact repetitions of **Person** and **Organization**, indicating these two classes are particularly represented and distinctive in the dataset. For instance, in one of the clusters, all nodes **Person** had two properties *wasIn* and *developed* indicating the fact that the person participated in an event and that she developed a kind of product, respectively. In another cluster, all **Person** nodes had the following properties: *bornIn*, *developed*, *birthDate*, *deathDate*, *diedIn*, and *wasIn*. After merging, the **Person** class will have all the labels of the second cluster of persons, which already includes those of the first. The selection of *mandatory* properties here must reflect at most the intersection between all common properties, otherwise making the knowledge graph inconsistent with respect to the schema.

For *impure clusters*, we need to distinguish whether the cluster is made up of correct labels or not. Based on this evaluation, two different steps are taken: if the designer recognises a certain hierarchy among all the labels, *creating hierarchies* is the successive

refinement step, meaning that the simulation has successfully recognised commonalities between different labels. This step may also involve the introduction of a new class acting as a lower class for the collected classes. For GraphBRAIN (i), more than 60% of the *impure clusters* are considered hierarchical clusters with respect to the schema, while for GraphBRAIN (ii), it is more than 25%. In this case, the identification of *mandatory* properties is also of great importance. Class hierarchies design consists of identifying inherited properties for each non-top class, as well as understanding *mandatory* ones for each of them. For instance, in one cluster, we have only the **Organization** and **Company** nodes, and we recognise that every company is an organization but not vice versa. For **Organization** the properties *produced* and *owned* were found while only *owned* was found in **Company** nodes. As schema designers, after introducing the hierarchy, I consider using the property *owned* as a *mandatory* property for justifying the existence of an organization.

After identifying *non-hierarchical clusters*, the designer is asked to refine them based on her understanding of the domain. More often than not, these clusters are not completely flawed but contain labels that, by chance or by data incompleteness, have only a few properties that are also shared by other labels. In GraphBRAIN (ii), the following labels were grouped into a cluster: **Book**, **Document**, **Year**, **Day**. One may easily recognise that there is space for two hierarchical classes: the first for **Book** and **Document**, and the other for **Year** and **Day**, perhaps introducing an upper class, what I referred to as **TemporalSpecification**. Basically, I used the *class splitting* conjunctively with *creating hierarchies*. If the designer is unsure about the distinctiveness of books over documents, she may also consider *merging*. This phase is of course the most expensive one, even though the fact of having an initial separation of data still represents an advantage, as if it were a bottom-up design approach.

The number of refinements is unpredictable, given the subjectivity of the designer and the different levels of granularity required, but the number of *pure clusters* and the *non-hierarchical clusters* represent a good estimation of how much work is needed, at least for a comparison between datasets.

7.3.4 Use Case Final Remarks

Summarizing, pure clusters occur quite often and can easily be merged together. After initialization, there are not even mandatory properties involved. For the impure clusters, I have hierarchical clusters, which deserve the introduction of mandatory properties, and the non-hierarchical ones, which require class splitting. The GraphBRAIN datasets look particularly useful to be refined through this approach, given their richness and the amount of data available. Schemas themselves are detailed and the instances possess many of the properties available in the conceptualization of their classes. As a consequence, the simulation in GraphBRAIN represents a good percentage of correlated classes (even though the variance is very high as in the example) and successive refinements do not involve severe data analysis. Though the high dimension of data and schemas are positively welcomed, they bring some general concerns to the approach. The process of extracting a specific piece of the graph makes the refinement dependent on the labels/properties seen, and some considerations may not hold for other subparts. The problem of subjectivity becomes even more subtle when different designers look at different parts of the graph. Both of these problems are mitigated when a good starting ontology is available, guiding the general behaviour of labels and properties, and a common background of the domain.

7.4 Evaluation on Other Datasets

Apart from the GraphBRAIN datasets, I tested the approach with excerpts of other standard datasets: the Twitter Neo4j graph[1] and the Lehigh University BenchMark (LUBM) dataset[2].

For Twitter, we have a very simple conceptualization of data, with very few labels, 7 distinct arc labels, and no hierarchies. Given the simplicity of the schema and the absence of hierarchies, I expect few or no refinements to this schema.

For LUBM, we have a rich schema defining concepts and roles in the university domain. Unfortunately, instances do not reflect the complexity of the schema: most classes do not have instances in the data and very few properties are available on them,

[1] https://towardsdatascience.com/using-neo4j-graph-database-to-analyze-twitter-data-6e3d38042af1

[2] https://swat.cse.lehigh.edu/projects/lubm/

most of them not even distinguishing (e.g., only *name*). On average, ∼20/30 distinct arc labels per excerpt are available. I expect from this dataset that the data-driven approach will not be as efficient as for GraphBRAIN since there will not be too many distinct edge labels for understanding generalization (specializations, resp.) among labels. In contrast, it may be the case that the approach highlights this inequality between the levels of detail of conceptualization and instances. Data has been extracted in a similar way as done for GraphBRAIN:

- for the Twitter dataset, we found thousands of users (almost) sharing the same features. For this reason, I selected (different) random samples of users keeping for each of them all his/her features. For each sample, I collected 30% of the users.
- for the LUBM dataset, I used the same strategy but took random samples of students and universities. About 40% of the instances have been extracted for each test.

For the Twitter dataset, the approach succeeded straightforwardly because the schema and instances are perfectly aligned, and the properties are fully distinguishable. This is shown by the fact that each cluster of nodes contains exactly one distinct label (e.g., **User**, **Tweet**, **Hashtag**). 8 labels are available in total and all the clusters are *pure* after schema initialization (cf. 7.2.1). Multiple runs of the same experiments did not lead to significant changes, showing that the conceptualization of the Twitter dataset, to date, does not need refinements.

Conversely, on the LUBM dataset, more considerations can be made. Albeit it correctly collects nodes to be put into hierarchies like the **Course** and **GraduateCourse** nodes, it also correlates labels that should be distinct, like **Person** and **University**. Different from GraphBRAIN, in which these non-hierarchical clusters appear by chance and are different, this behaviour in LUBM is the result of a noisy dataset because, even though having a rich conceptualization, data does not reflect this complexity. For instance, many universities are just identified by their name, and for students, it is the same situation. The obvious result is that many universities simulate users, and vice versa. This advocates that students should be identified by more specific features like ID assigned by the university, passed exams and so on. Hence, the experiment on this dataset demonstrates a flaw in the instances, represented in a shallow way, as well as the introduction of *mandatory* distinguishable properties in the schema must be con-

sidered. Anyway, the *pure clusters* represented ∼80%, making the refinement process not particularly demanding either.

Apart from the above considerations, I also require this approach to establish how much the initial conceptualization fits with respect to the instances. I depicted the similarity between groups of labels and how much these labels reflect the ground truth (via the available schema). For each collection of labels (i.e., cluster), I measure how different they are. I presume that in each cluster there will be a predominant label, but the metrics do not depend on this assumption. For each cluster, I compute, for each label l_i, its frequency f_i within the cluster, take the maximum and divide it by the cluster cardinality following the formula:

$$\text{sim}_{\text{cluster}} = \frac{\max(f_1, f_2, ..., f_k)}{\sum_{i=1}^{k} f_i}.$$

This metric gives an estimation of the number of refinements but does not measure the matching between the schema initialization and the correct schema. To this extent, I need to measure, after grouping nodes by the simulation function, how many labels of the collected nodes are not grouped non-hierarchically. There is no standard metric in the literature for this task but I opted for a cluster conformance metric providing a ratio between the number of labels outside the longest hierarchy identified in the set of labels and the number of total labels and then subtract this value from 1.

Suppose we have a cluster of six labels A, B, C, D, E, F with $C \sqsubseteq B \sqsubseteq A$ and $E \sqsubseteq D$, then the distance with respect to the ontology is $1 - \frac{3}{6} = \frac{1}{2}$ because of the 3 distinct labels $\{D, E, F\}$ outside the longest hierarchy $\{A, B, C\}$. It is reasonable to assume that there is a predominant hierarchy in the cluster. The label *String* is not taken into account, while for *pure clusters* this ratio is always 1. When computing averages, I neglected them.

Table 7.2 confirms our expectations for the simple and well-defined Twitter dataset in which no refinements are needed since the average $\text{sim}_{\text{cluster}}$ is 1 while manifesting the need for refinements for both LUBM and GraphBRAIN. Specifically, the presence of distinct labels may be in favour of hierarchies but we need to distinguish whether this is true in the ground truth (the initial schema). The Twitter graph turned out to be irrelevant for this scope since every cluster was pure, demonstrating full compliance with the schema.

Although LUBM has higher cluster similarity than GraphBRAIN, its conformance

Table 7.2: Difference and conformance evaluation

Dataset	Avg Cluster Similarity	Avg Cluster Conformance
GraphBRAIN (i)	0.91	0.48
GraphBRAIN (ii)	0.89	0.86
Twitter	1.00	1.00
LUBM	0.97	0.53

distance is not really high, meaning that a high percentage of collected labels were not supposed to be linked in the schema. GraphBRAIN, although showing much variance, seems to suffer from the same problem of LUBM in some of its parts, but some excerpts demonstrate how profitable this approach would be in cases where comprehensive schema meets expressive data. However, these numbers tell us only an initial estimation of the effort for refinements. It is necessary to remember that many steps depend on data and prior knowledge, and a smaller average cluster conformance does not necessarily mean to refine in fewer steps.

7.5 Discussion

Automatic schema evaluation is not a popular line of research, although many contexts may benefit from that. In this work, a semi-automatic strategy differentiates from the methodological and evaluation point of view from other approaches. Lemaitre et al. [283] introduced the idea of concept mapping in order to sort or correlate semantics of DB constraints. The proposed strategy relies only on the structural properties of instances, so *data-based* and the designer is at support in deciding which of the three basic operations needs to be performed. The evaluation has to consider also the specific schema and data compliance.

Chapter 8

Graph Logic Programming

Given the settings of graph databases and schemas, in this chapter, there will be introduced some interesting new methodologies and use cases demonstrating how the model in first-order logic graph-related tasks. Three graph-based tasks will be explored: graph reachability (which can be considered an example of a more general class of problems), instance matching and argumentation mining.

8.1 Graph Reachability

Graph reachability is the task of understanding whether two distinct points in a graph are interconnected by arcs to which in general a semantic is attached. Reachability has plenty of applications, ranging from motion planning to routing. Improving reachability requires structural knowledge of relations so as to avoid the complexity of traditional depth-first and breadth-first strategies, implemented in logic languages. In some contexts, graphs are enriched with their schema definitions establishing domain and range for every arc. The introduction of a schema-aware formalization for guiding the search may result in a sensitive improvement by cutting out unuseful paths and prioritising those that, in principle, reach the target earlier. In this chapter, I propose a strategy to automatically exclude and sort certain graph paths by exploiting the higher-level conceptualization of instances. The aim is to obtain a new first-order logic reformulation of the graph reachability scenario, capable of improving the traditional algorithms in terms of time, space requirements, and number of backtracks. The experiments exhibit the expected advantages of the approach in reducing the number of

backtracks during the search strategy, resulting in saving time and space as well.

8.1.1 Context

I define or redefine here some concepts in order to provide a formalization that will be helpful throughout this section.

Definition 1 A graph $\mathcal{G}$ is a pair ($\mathcal{V}$, $\mathcal{E}$) in which $\mathcal{V}$ represents the set of vertices and $\mathcal{E}$ is a relation of *incidence*, associating vertices with each other. Each edge associates a vertex called the *start* with a vertex called the *end* of the edge. It can be indicated as $s \xrightarrow{e} o$, standing for an edge named e between the start s and the end o.

Given a graph, we are able to define the *reachability* in terms of the *graph* definition:

Definition 2 Given a graph ($\mathcal{V}$, $\mathcal{E}$), a node o is reachable from a node s iff $\exists$ a path $p = e_0, e_1, ..., e_{k-1}$ s.t. $s \xrightarrow{e_0} s_1 \xrightarrow{e_1} s_2 \xrightarrow{\cdots} s_{k-1} \xrightarrow{e_{k-1}} o$, and we call k the cardinality of the set $\{e_0, e_1, ..., e_{k-1}\}$ one of the lengths between s and o. If a path p does not exist, we say that o is not reachable from s and the distance between s and o is ∞.

In general, we require $s_0, s_1, ..., s_{k-1}$ to be distinct to avoid useless loops, but this is negligible for the purpose of the work.

Definition 3 Given an alphabet Σ, an LPG graph $\mathcal{G}$ is a 4-tuple ($\mathcal{V}$, $\mathcal{E}$, $\mathcal{L}$, $\mathcal{P}$) in which $\mathcal{V}$ and $\mathcal{E}$ are the same as for definition 8.1.1, $\mathcal{L}$ is a labelling function $(\mathcal{V} \cup \mathcal{E}) \rightarrow 2^{\Sigma^*}$ associating to each node (resp. edge) a set of labels generated from the alphabet, and $\mathcal{P}$ is a named property function $(\mathcal{V} \cup \mathcal{E}) \rightarrow (\epsilon \cup \Sigma^+)$ associating the value of a property to each node (resp. edge).

Definition 4 A schema graph is a 4-tuple ($\mathcal{E}$, $\mathcal{R}$, $\mathcal{P}$, $\mathcal{S}$). $\mathcal{E}$ is the set of *entities*. A special entity **Thing** is always available. $\mathcal{R}$ is the set of *relationships*, a set of labelled functions $\mathcal{E} \rightarrow 2^E$, expressing dependencies among instances of the involved entities. It can be indicated as $s \xrightarrow{e} o$, standing for a relationship named e between the start entity s and the end entity o. Given an alphabet Σ, $\mathcal{P}$ is a named property function $(\mathcal{E} \cup \mathcal{R}) \rightarrow \{Int, String, Date\}$ associating to entities (resp. relationships) properties with their associated domains. Without losing generality, we can refer to only integer, string and date values. $\mathcal{S}$ is the *subclass* relation $\mathcal{E} \rightarrow \mathcal{E}$ indicating that the first entity

is a specialization of the latter. Multiple inheritance is not supported; hence, $\mathcal{S}$ creates a tree structure, having entity **Thing** as root. Given an e $\in \mathcal{E}$, we define $\Gamma^0(e) = e$, $\Gamma^1(e) = \mathcal{S}(e)$, and in general $\Gamma^n(e) = \mathcal{S}(\Gamma^{n-1}(e))$ until *Thing* is reached. We define $Super(e) = \bigcup \Gamma^n(e)$ as the fix point operator of Γ applied to e to indicate the labels of e and all its super-entities.
Slightly changing Definition 8.1.1, we can define reachability for graph schemas.

Definition 5 Given a graph schema ($\mathcal{E}$, $\mathcal{R}$, $\mathcal{P}$, $\mathcal{S}$), an entity o is reachable from an entity s iff $\exists$ a path $p = e_0, e_1, ..., e_{k-1}$ s.t. $s = s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} s_2 \xrightarrow{\cdots} s_{k-1} \xrightarrow{e_{k-1}} s_k = o$ s.t. $\forall i \in \{0, k-1\}$ $Super(s_i) \xrightarrow{e_i} Super(s_{i+1}) \in \mathcal{R}$ and k is one of the lengths between s and o.

Differently from Definition 8.1.1, schemas take into account hierarchies. Given a graph, we need to define when a graph is consistent with respect to a graph schema.

Definition 6 Given a (labelled property) graph ($\mathcal{V}$, $\mathcal{E}$, $\mathcal{L}$, $\mathcal{K}$) and a graph schema ($\mathcal{C}$, $\mathcal{R}$, $\mathcal{P}$, $\mathcal{S}$), a graph is compliant with a graph schema iff:

- $\forall v \in \mathcal{V}$ $L(u) \subset \mathcal{C}$ (i)
- $\forall e \in \mathcal{E}$ $e \subset \mathcal{R}$ (ii)
- $\forall v \in \mathcal{V}$ $\forall k \in \mathcal{K}(v)$ s.t. $\mathcal{K}(v) \neq \epsilon \implies \exists p \in \mathcal{P}$ $\exists l \in \mathcal{L}(v)$ s.t. $p(l) \neq \epsilon \wedge$ k and p are equally named (iii)
- $\forall e \in \mathcal{E}$ $\forall k \in \mathcal{K}(e)$ s.t. $\mathcal{K}(e) \neq \epsilon \implies \exists p \in \mathcal{P}$ $\exists r \in \mathcal{R}$ s.t. $p(l) \neq \epsilon \wedge$ k and p are equally named (iv)
- $\forall s \xrightarrow{e} o \in \mathcal{E} \implies \exists r \in \mathcal{R}$ s.t. $(ls, lo) \in r \wedge ls \in Super(s) \wedge lo \in Super(o) \wedge$ e and r are equally named (v).

In short, conditions (i) and (ii) state that labels of nodes and edges *must* exist in the schema. Conditions (iii) and (iv) state that every property of a node (resp. edge) in the graph *must* be presented in the schema, and vice versa every property in the schema *can* be present in the graph. Condition (v) states that for each edge in the graph, there *must* be a relationship in the schema defined for the labels of the connected nodes, or one of their respective super-entities. Conversely, if a relationship exists in the schema it *may* be represented in the graph.

Listing 8.1: Extract of "User"-related facts.

```
entity(agent).
subclassOf(agent, entity).
entity(person).
subclassOf(person, agent).
entity(personUser).
subclassOf(personUser, person).
arc(person, collection).
arc(person, event).
arc(person, organization).
arc(person, place).
...
```

The basis of the following considerations starts from this result:

Proposition 1 Given a graph $\mathcal{G}$ compliant with a graph schema $\mathcal{S} = (\mathcal{E}, \mathcal{R}, \mathcal{P}, \mathcal{S})$, a node o is reachable from node s only if the label of o is reachable from the label of s in the schema.

Proof Suppose $\exists$ a path $p = s \xrightarrow{e_0} s_1 \xrightarrow{e_1} s_2 \xrightarrow{\cdots} s_{k-1} \xrightarrow{e_{k-1}} s_k$ then, from (v) of Definition 8.1.1, $\exists\ c_0 \xrightarrow{r_0} c_1 \xrightarrow{r_1} c_2 \xrightarrow{\cdots} c_{k-1} \xrightarrow{r_{k-1}} c_k \in \mathcal{R}$ s.t. $c_0 \in Super(s)$, $c_1 \in Super(s_1)$, ..., $c_k \in Super(s_k) \implies c_k$ is reachable from c_0.

8.1.2 Problem Reformulation

Here we propose how to solve the problem in a more efficient way by resorting to the lists in the **arcs/2** predicates. We frame the problem under the condition of having a graph and a graph schema expressed as first-order logic clauses. In general, we deal with Horn clauses [284], and the information about the graph and its schema is expressed as *facts*, a specific Horn clause with no negated terms. As previously said, we prefer a structural representation of the schema, focusing on entities, relationships and the subclass relation. Specifically, we have the **entity/1**, **subclassOf/2**, and **arc/2** predicates. We report in Listing 8.2 an excerpt of predicates involving the entity **Person**. It is not a top entity since its super-entity is **Agent** which, on the other hand, is a top entity because is a child of *Thing*. **Person** has one sub-class (**PersonUser**), and some arcs with entities like **Event**, **Organization** and **Place**. Remind that all the arcs of the super-entity **Agent** are also arcs for **Person**.

Listing 8.2: Excerpts of facts.

```
node(1).
node(2).
node(3).
...
label(1, a).
label(2, b).
label(3, c).
...
arcs(1, [2]).
arcs(2, [2,1]).
arcs(3, [4]).
```

For each distinct pair of entities in the graph schema knowledge base, it must be computed the minimum distance between the pair. Our resolution, algorithmically described in Listing 8.3, has been implemented in *Answer Set Programming* and is inspired by the distance-vector routing algorithm [285] that takes as input the set of pair of entities and returns a function specifying for each pair its distance. The computation of all distances between entities can be regarded as a preprocessing step since, unless ontological changes, these do not need to be recomputed when reachability involves different nodes. Ontological changes happen when the domains of use, users and/or management or law requirements change which is, fortunately, not frequent. After this preprocessing step, we are interested in filtering only pertaining distances, that are the ones involving the target node. The new graph navigation *must* take into account distances between labels. Given proposition 8.1.1, we can exploit reachability among labels of nodes before navigating the graph. This may, in principle, (almost) always represent a benefit for the computation since labels are extremely fewer than nodes and the resorting (with pruning) of paths should increase the likelihood of finding the target node in a shorter time. The resorting of nodes prunes path for which the distance of labels is ∞, and the remaining (feasible) paths will be sorted according to the distance between labels. For instance, given the labels S, A, B and T (target), with $distance(A, T) = 1$ and $distance(B, T) = 2$, if a node labelled S has connections with A and B, it will prefer the path over A since, in principle, it will reach a node equally labelled the target one shortly. How often this approximation is reliable will be a matter of discussion and it is, in fact, dependent on the dataset. Instances are represented with two predicates: **node/2** and **arcs/2** where the first one represents

Listing 8.3: Algorithm for distance computation

```
Input: Set<Pair> pairs
Output: Pair --> Int distance
compute_distances(pairs):
    for each pair in pairs:
        distance(pair) := find_minimum_distance(pair, [])
    return distance

Input: Pair pair, List<Int> visited
Output: Int min_distance
find_minimum_distance(pair, visited):
    min_distance := infty
    if pair[0] = pair[1]
        return 0
    else
        for each y in arc(pair[0], y) and y not in visited:
            distance :=
            find_minimum_distance(y, pair[1], [y | visited])+1
            if distance < min_distance
                min_distance := distance
        return min_distance
```

the id of the node associated with its label, while the latter all the connections between ids. The second term of the predicate **arcs** is the list of adjacent nodes, and it is the list that needs to be sorted. We will call **improved_arcs** the predicate derived by the sorting of connections with the label distances, and the solution is shown in Listing 8.4, as well as the improved reachability version.

Full implementation of the improved graph connectivity is available in the Appendix (A.1).

8.1.3 Evaluation

We tested our modified approach over two datasets: GraphBRAIN and the Twitter LPG dataset[1]. These two datasets have a very different complexity in the graph schema. The GB-selected ontology represents more than 140 entities and more than 90 arcs between them. On the other hand, the Twitter graph schema is composed of 6 entities and 13 arcs. Given the difference in their granularities, we can compare how much advantage is gained with respect to the level of detail of schemas. For GB, we tested several excerpts of the dataset. The rationale was to start from a set of representative nodes (ranked with PageRank), and then use a spreading activation function to grasp

[1]https://github.com/neo4j-graph-examples/twitter-v2

Listing 8.4: Sorting arcs and improved reachability

```
distance_target(X, D) :- distance(T, X, D), target(T).

compare_order(<, X, Y) :- node(X, L1), node(Y, L2), OX < OY,
     distance_target(L1, OX), distance_target(L2, OY).
compare_order(>, X, Y) :- node(X, L1), node(Y, L2), OX >= OY,
    distance_target(L2, OY), distance_target(L1, OX).

has_distance(X) :- node(X, L), distance_target(L, _).

sort_arcs :- forall(arcs(Node, List),
        (
            include(has_distance, List, FilteredList),
            predsort(compare_order, FilteredList, SortedList),
            assertz(improved_arcs(Node, SortedList))
        )
    ).

%improved reachability
improved_reach(X, X).
improved_reach(X, Y) :- !, improved_reach(X, Y, []).
improved_reach(X, Y, _) :- improved_arcs(X, L), member(Y, L).
improved_reach(X, Y, Visited) :- improved_arcs(X, L),
    member(Z, L), Z \= Y, \+ member(Z, Visited),
    improved_reach(Z, Y, [Z|Visited]), improved_arcs(X, L).
```

connected nodes from these central nodes. We extracted ∼5k nodes and ∼5k arcs per execution for GB, and ∼700 nodes and ∼700 arcs per execution for Twitter. We report here only results for a specific execution, since multiple runs of the same experiment did not lead to significant changes. Evaluation has been conducted by comparing the *reachability* before and after optimization, and by measuring three percentages: number of executions where the improved version led to better performance, saved execution time and number of saved backtracks. The first percentage tells us how frequent is the advantage of using the improved version, the second gives us a glance at the value of the optimization, while the latter one is an objective measure, non-dependant on the specific performance of the algorithm and the machine used. Since each excerpt started from a central node, we computed *reachability* from all the nodes to the central one. We tested GB and Twitter using SWI-Prolog[2]. All experiments have been conducted on an Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz-1.50 GHz processor with 16GB of RAM.

[2]https://www.swi-prolog.org/

Table 8.1: Results

Dataset	% Improved Time	% Saved Time	% Saved Backtracking
GraphBRAIN	**77**	**75.8**	53.7
Twitter	68.5	36.5	**59.2**

Listing 8.5: Date library in Prolog

```
measure_execution_time(Query, Time) :-
    get_time(Start),           % Get time before the query
    call(Query),               % Execute the query
    get_time(End),             % Get time after the query
    Time is End - Start.       % Calculate the elapsed time
```

Table 8.1 shows the results for an instance of execution.

The results demonstrated the usefulness of the approach, especially in the GB setting, where the graph schema is well-detailed and pruned to a greater extent. The second column tells us that, in 77% of reachability computations, the improved version reduced the needed time to compute to fulfil the task. On average, the % saved time, computed as

$$\frac{time_{reachability} - time_{improved\_reachability}}{time_{reachability}} \cdot 100$$

is more than 75, also reaching peaks of 99% for some cases. In terms of orders, the magnitude of saved time is in the order of 100 i.e., the improved version is 100x faster than the *blind* version on average. Following our expectations, the same results do not happen for the Twitter dataset. However, it is still valuable to use a graph schema-aware reachability in more than 68% of cases. Surprisingly, the percentage of saved backtracking is even higher than GB, even if the saved time is much lower. We have a guess for this behaviour: since there is high connectivity among nodes in Twitter and there are several arcs connecting nodes of the same class, our schema-aware reachability avoids useless loops visiting nodes of the same class and, at a certain point, still reaches the target node.

Times have been computed with the *date* library of Prolog, considering the difference between the absolute time before and after the call to the *reachability* query, as shown in Listing 8.5.

8.2 Argumentation with Optimization

In the argumentation field, several semantics have been proposed to identify subsets of arguments showing properties based on self-consistency or defence against attackers. Schemas introduce in KBs constraints that cannot be violated and for which the problem can be tackled by argumentation-based semantics. One of the first examples that come to mind, mentioned in [286], is the presence of two mutually excluded facts like the presence of more than one arc referring to a functional relationship (e.g. *bornIn*). Reasoning in Abstract Argumentation is the activity of understanding subsets of arguments that resolve conflicts or can be used as a justification for a thesis. Several semantics have been proposed so far to identify subsets of arguments showing properties based on self-consistency or defence against attackers. Real-world scenarios and peculiarities of many problems gave rise to the introduction of more complex frameworks that also specify degrees of certainty of arguments and attacks. However, semantics barely take into account the fuzziness of these "beliefs" properly. In this work, we propose a strategy to compute arguments' beliefs based on the initial degrees of certainty of arguments and propose a new semantic, *most-probable*, that, starting from a target argument we aim to justify, captures the set of admissible arguments containing the target argument which maximizes the minimum belief of the whole set. The introduction of this new semantic can facilitate the understanding of the consequences of supporting a claim, neglecting those that may result in low belief.

8.2.1 Background

Although this is not intended to be a comprehensive review of the literature in the argumentation field, we provide basic definitions and concepts for this work. Definitions mainly have roots from the first Dung's formalization and the proposed basic semantics.

Definition 1 An *Argumentation Framework (AF)* is a pair $\langle Args, Att \rangle$ where $Args$ is a (finite) set of arguments and $Att \subseteq Args \times Args$ the attack relationship. If $\langle A, B \rangle \in Att$ we say that argument A *attacks* argument B (or that A is an attacker of B). For each $A \in Args$, we denote as $A^- = \{ B \in Args : \langle B, A \rangle \in Att \}$ the set of all *attackers* of A, and as $A^+ = \{ B \in Args : \langle A, B \rangle \in Att \}$ the set of all *attacked* by A. In this scope, we neglect self-attacking arguments, so $A \in Args \implies A \notin A^- \ (A^+)$.

The concept of attack can also be extended to a set of arguments. Given $S \subseteq Args$, $A \in Args$ *attacks* S iff *exists* $B \in S$ such that A attacks B. Conversely, a set $S \subseteq Args$ attacks $A \in Args$ iff $\exists B \in S$ such that B attacks A. Figure 8.1 shows an example of an argumentation graph.

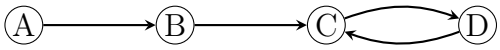


Figure 8.1: An example of an Argumentation Framework with arguments A, B, C and D.

Following Li et al. [287], we provide the following definition.

Definition 2 Let an $AF\ G = \langle Args, Att \rangle$, a *Probabilistic Argumentation Framework (PAF)* $\langle Args, Att, PArgs, PAtt \rangle$ is a 4-tuple in which $Args$ and Att are defined as above, $PArgs : Args \rightarrow]0, 1]$ and $PAtt : Att \rightarrow]0, 1]$ functions respectively indicating the likelihood of arguments and attacks.

It is often the case that the terms "probability" and "belief" are confused. They both represent a degree of certainty but have different meanings and implications, as indicated by Fagin et al. [288]. Figure 8.2 shows an argumentation graph with probabilities attached to arguments and attacks.

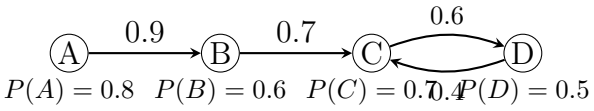


Figure 8.2: An example of a Probabilistic Argumentation Framework.

Slightly modifying Definition 1, we can generalise the probabilistic framework with a generic *weighted* framework. From Bistarelli et al. [289] we provide the following definition.

Definition 3 Let an $AF\ G\langle Args, Att \rangle$, a *Weighted Argumentation Framework* $\langle Args, Att, WAtt$ is a triple in which $Args$ and Att are defined as above, and $WAtt : Att \rightarrow \mathbb{R}_{>}$ a function indicating the weight of attacks.

Differently from probabilities, the meaning of weights does not reflect how probable is the attack, but how *much* the arguments attack. Figure 8.3 shows an argumentation graph with weights attached to attacks.

The Argumentation Framework and its variants can be enriched with the *support* relationship, the opposite of the attack. In some contexts, it is advantageous to add

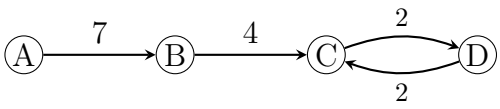


Figure 8.3: An example of a Weighted Argumentation Framework.

support relationships, but these considerations are beyond the scope of this work.

Given an AF, we are interested in collecting sets of arguments that can be generally considered *acceptable*. The condition of being acceptable depends on the *semantics*.

Definition 4 Let an $AF\ G = \langle Args, Att \rangle$ be an AF, an *extension-based semantics* S associates AF with a subset of 2^{Args}, denoted by Every subset of 2^{Args} can be considered a distinct semantics (even $\emptyset$), but only some of them are convenient for acceptability. The basic condition for a set of arguments to be generally acceptable is to show the *conflict-free* condition;

Definition 6 Let an $AF\ G = \langle Args, Att \rangle$ be an AF, a set of arguments $S \subseteq Args$ is *conflict-free* iff $\nexists \langle A, B \rangle \in S$ such that $\langle A, B \rangle \in Att$. The set of all conflict-free extensions is indicated as cf(G).

Referring to the example in Figure 8.1, in Figure 8.4 two ways of grouping arguments into conflict-free sets are shown.

Figure 8.4: Two instances of the same argumentation graph. In the left picture, A is put together with C, and hence B and D can also be collected; in the right picture, A is put together with D, and then B and C must be separated.

This semantics guarantees to obtain groups of arguments at least *consistent* with each other. Many semantics have a hierarchical structure, from the least restrictive to the most. The conflict-free is the minimal result for acceptability. More interesting semantics start from the conflict-free one. Note that $\emptyset$ is always conflict-free, and, with the assumption of non-self-attacking arguments, singleton sets are conflict-free as well, but this result is not practically interesting.

For distinguishing some semantics it is necessary to define the concept of *defense*.

Definition 5 Let an $AF\ G = \langle Args, Att \rangle$, $S \subseteq Args$, and $\alpha \in Args$, S *defends* α iff $\forall \gamma \in Args\ \langle \gamma, \alpha \rangle \in Att \implies \exists B \in S$ such that $\langle B, \gamma \rangle \in Att$.

Definition 7 Let an $AF\ G = \langle Args, Att \rangle$ be an AF, a set of arguments $S \subseteq Args$ is *admissible* iff S is conflict-free and $\forall \gamma \in Args\, \gamma$ attacks $S \implies S$ attacks γ. In other words, S defends every $A \in S$. When this happens, we say that S is able to defend itself. The set of all admissible extensions is indicated as adm(G). In Figure 8.4 three pairs of arguments are conflict-free: $\{A, C\}$, $\{B, D\}$, $\{A, D\}$, but $\{B, D\}$ is not admissible since A attacks B but neither B nor D attacks A; hence, it cannot defend itself. Figure 8.5 shows the two possible pairs of arguments, white nodes cannot be part of any admissible set.

Figure 8.5: Two instances of the same argumentation graph. In the left picture, A and C are admissible, but not B and D, even alone; in the right picture, A and D are admissible, and then B and C cannot be part of any admissible set.

In general, we are not only interested in admissibility but also in including as many arguments as possible in order to make the discovery process more interesting.

Definition 8 Let an $AF\ G = \langle Args, Att \rangle$ be an AF, a set of arguments $S \subseteq Args$ is a *complete* extension iff S is admissible and $\forall \alpha \in Args\ S$ defends $\alpha \implies \alpha \in S$. The set of all complete extensions is indicated as comp(G).

Completeness of semantics acts as a spreading of the defence relationship. Starting from a singleton, in order for the set to be complete, it must include all the arguments it defends, and then progressively add all the others defended by the ones already included in the set.

Having collected all complete extensions, we can collect all the arguments available in every complete extension by intersection.

We define from Ferilli et al. [290] *grounded* semantics.

Definition 9 Let an $AF\ G = \langle Args, Att \rangle$ be an AF, a set of arguments $S \subseteq Args$ is a *grounded* extension iff S is complete and $\nexists T \in$ comp(G) such that $T \subset S$. The grounded extension is the minimal set of arguments belonging to the complete extensions. It is unique and always exists.

In general scenarios, it may be preferable to select extensions that guarantee as many arguments as possible. To this extent, we can define *preferred* extensions.

Definition 10 Let an $AF\ G = \langle Args, Att \rangle$ be an AF, a set of arguments $S \subseteq Args$ is a *preferred* extension iff S is admissible and $\nexists T \in \text{adm(G)}$ such that $T \subset S$. The set of all preferred extensions is indicated as pref(G).

Finally, when computing extensions, it happens that not all the arguments outside preferred extensions are attacked. These are the so-called "undecided arguments". *Stable* extensions split the set of arguments between arguments in the extension and attacked arguments.

Definition 11 Let an $AF\ G = \langle Args, Att \rangle$ be an AF, a set of arguments $S \subseteq Args$ is a *stable* extension iff S is conflict-free and $\forall \alpha \in Args\ \alpha \notin S \implies S$ attacks α. The set of all stable extensions is indicated as stb(G).

8.2.2 Probabilistic Update and Most-Probable Extension

Given the variety of argumentation models and semantics, we define here our settings for the work. As previously said, research on argumentation semantics under uncertainty is still an underrepresented line of research. We start from a small variety of the PAF defined above. Specifically, we suppose probabilities only on arguments. Probabilities come under the assumption of *non-additivity*, meaning that it is not impossible that $\Delta \not\models \alpha$ and $\Delta \not\models \neg\alpha$. Attacks always trigger with the same "intensity". Probabilities of arguments may come from several external sources. In general, statistical approaches come into play when providing an initial estimation of a phenomenon. In this setting, probabilities of arguments do not take into account correlation with other arguments. Albeit this assumption may not hold in real scenarios, it facilitates an initial approximation of probabilities, which would be extremely demanding otherwise since statistical information of events can barely take into account all surrounding conditions.

Formally, we can define our version of PAF, namely *Simplified PAF (SPAF)*.

Definition 12 Let an $AF\ G = \langle Args, Att \rangle$, a *Simplified Probabilistic Argumentation Framework (SPAF)* $\langle Args, Att, PArgs \rangle$ is a triple in which $Args$ and Att are defined as above, $PArgs : Args \rightarrow]0, 1]$, a function indicating the likelihood of arguments.

Figure 8.6 shows an example of SPAF.

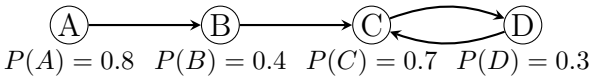


Figure 8.6: An example of a Simplified PAF.

Note that probabilities of arguments cannot be compared with Kolmogorov Axioms [291] because we do not know in advance whether two arguments are independent. Even C and D in the example above cannot be considered mutually excluded given possible flaws in the statistical process. The process of attack creation needs an expert in the field who, with the help of prior knowledge, can design a complete SPAF. Conversely, retrieving probabilities of attacks may present some unsound scientific processes or require too accurate domain knowledge.

Probabilities of arguments are "blind" with respect to other arguments and attacks. For this reason, we need to update the probabilities of arguments, based on attacks and the initial probability of arguments. As a general rule, we expect the updated probability inversely proportional to the probability of the attacker and the number of attackers.

The update should consider using a hyperparameter $\alpha \in]0, 1]$ to establish the pace at which values change. The hyperparameter is an indicator of how much attacks should impact the acceptability of attacked arguments.

The final formula will be

$$P'(A) = P(A) \cdot \prod_{\gamma \in Args \, \langle \gamma, A \rangle \in Att} 1 - \alpha \cdot P(\gamma)$$

where $P'(A)$ represents the probability of A after being attacked by all its attackers. The formula presents nice properties but has also some limitations or assumptions. Nice properties are:

- multiple attacks influence exponentially the updated belief, for this reason, a low α may bring less severe changes.
- the product $1 - \alpha \cdot P(\gamma)$ is always between 0 and 1; hence, the final result is still a probability.
- the initial belief of an argument is an upper bound for the update since multiplications of factors between 0 and 1 only reduce the number.

- many weak (low P(γ)) attackers are less influential than few strong (high P(γ)) attackers.

Some limitations are:

- if an attacker has probability 1, only with $alpha = 1$ the attack has the maximum effect (the updated probability of the attacked node is 0).
- the function is nonlinear, meaning that small changes have large effects. This is mitigated by the fact that we are more interested in relative probabilities among arguments rather than absolute numbers.
- it is assumed attacks are independent, which is in general false but neglected in many contexts given the complexity of the dependencies.

We report in Figure 8.7 two argumentation graphs, the first one after updating probabilities with $\alpha = 0.5$ and the second with $\alpha = 0.9$.

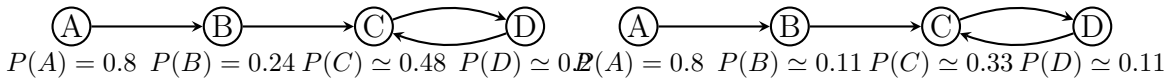


Figure 8.7: Two instances of probabilistic update in argumentation graph. In the left picture, $\alpha = 0.5$; in the right one, $\alpha = 0.9$.

Note that the whole discussion with probabilities does not change if we deal with generic "degrees of belief". We chose probability values given their statistical interpretation and the availability of statistical data about phenomena.

With this setting, it can be considered the possibility of performing an iteration process over probabilities. In this case, the iterative process of probability update will be the following:

$$P^n(A) = \begin{cases} P(A) & \text{if } n = 0, \\ P^{n-1}(A) \cdot U^n(A) & otherwise \end{cases}$$

where

$$U^i(A) = P^{i-1}(A) \cdot \prod_{\gamma \in Args \ \langle \gamma, A \rangle \in Att} 1 - \alpha \cdot P^{i-1}(\gamma).$$

However, we are currently interested in relative proportions among probabilities. The iterative process keeps proportions without adding further information. For this reason, in this work, we only make use of a single update step.

Given this argumentation framework, we are ready to define the *most-probable* extension. Differently from other semantics, we define this semantics starting from an argument we call "target". We need first to specify how we consider the probability of a set. Given $\{a_1, a_2, ..., a_n\} \subseteq Args$, $P(\{a_1, a_2, ..., a_n\}) = \min\{P(a_1), P(a_2), ..., P(a_n)\}$.

Definition 13 Let $\langle Args, Att, PArgs\rangle$ G a *Simplified Probabilistic Argumentation Framework* and $t \in Args$, a set of arguments $S \subseteq Args$ is a *most-probable* extension for t, indicated as $S \in \text{most-prob}_t(G)$ iff $t \in S$, S is admissible and $\nexists Y \subset S$ such that:

- $t \in Y$
- Y is admissible
- $P(Y \setminus \{t\}) > P(S \setminus \{t\})$.

and $\nexists Z \subseteq Args$ such that:

- $S \subset Z$
- Z is admissible
- $P(Z \setminus \{t\}) = P(S \setminus \{t\})$.

In other words, S is *most-probable* with respect to t iff it is admissible, includes t, including any other argument will make the set non-admissible or reduce the minimum probability of the set, and removing any argument will not increase the minimum probability while keeping admissibility. By construction, the target argument t always belongs to most-probable sets, and its probability is not evaluated when computing the minimum, saving us from a *paradox of certainty*. Suppose $P(t) \simeq 1$, then it is likely that the singleton $\{t\}$ is the only most-probable set since adding any other arguments would probably reduce the minimum probability.

Proposition 1 There is no unique solution for the most-probable extension.

Proof by construction, suppose G a SPAF with $Args = \{a_1, a_2, ..., a_n, s_0, s_1, t\}$ where $\{a_1, a_2, ..., a_n, t\}$ are not attacked by anyone, and s_0, s_1 attack each other. Then, both $\{a_1, a_2, ..., a_n, s_0, t\}$ and $\{a_1, a_2, ..., a_n, s_1, t\}$ are $\text{most-prob}_t(G)$.
A variant of this semantics can be easily considered if, instead of getting the most

probable admissible set, we are interested only in admissible sets showing at least a probability of δ.

Definition 14 Let $\langle Args, Att, PArgs \rangle$ G a *Simplified Probabilistic Argumentation Framework*, $t \in Args$ and $\delta \in \mathbb{R}_{>}$, a set of arguments $S \subseteq Args$ is a δ-*most-probable* extension with respect to t iff S is most-probable with respect to t and $P(S \setminus \{t\}) >= \delta$ Note that $\delta = 0$ would reduce *threshold most-probable* to most-probable.

In most cases, the variant version may be more advantageous in order to not only optimise as much as possible the likelihood of extracted arguments but also to accept (resp. restrict) the result set. This extension should be used as a further step after *most-probable*. Given the probability of one of the most-probable, any threshold above would result in having the set, while a lower threshold provides additional information on other sets.

We report in Figure 8.8 a more complex example of SPAF (after probability updates) in which green nodes represent one of its *most-probable* extensions with respect to D.

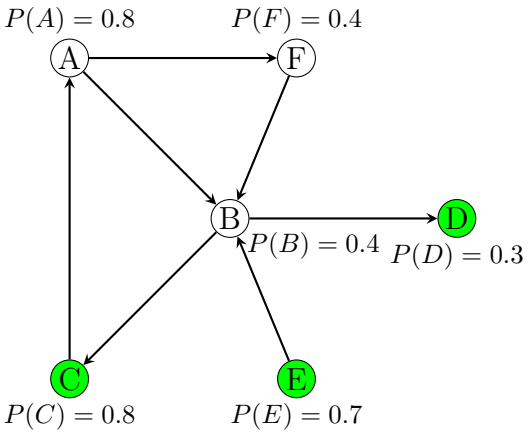


Figure 8.8: Most-probable extension with respect to D of a SPAF.

Note that F belongs to an admissible set in which there is also D but it reduced minimum probability. In a threshold extension with $\delta < 0.4$, F is included.

8.2.3 Implementation

For implementation, we developed this probabilistic argumentation framework in a platform for argument reasoning called *ARGuing Using Enhanced Reasoning (Arguer)*. It is an interactive Prolog system for argument reasoning in which the most common

frameworks and semantics have been developed. The system makes use of the YAP[3] compiler. Experiments of the proposed work have been performed on an Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz-1.50 GHz processor with 16GB of RAM.

8.2.4 Arguer

Arguer is a generic argumentation reasoning platform tailored for domain experts who can provide arguments in the form of propositional logic. The formalism to express arguments, attacks, or other components (e.g. weights) is predetermined. The knowledge about arguments and attacks can be fetched from a Prolog-like text file or by constructing facts directly from the interface. Based on the selected argumentation framework, several extensions can be computed.
Arguer provides the following argumentation frameworks:

- Abstract Argumentation Framework (AF)
- Value-Based AF
- Bipolar AF
- Weighted AF
- Bipolar Weighted AF
- Simplified Probabilistic AF

Figure 8.9 shows the main dialogue of Arguer.
Now we describe, for each framework, the formalism and the possible operations.

Abstract Argumentation Framework (AF) To express arguments and attacks, the predicates `argument/1` and `attack/2` are used. The first one specifies the name of the argument, and the second the relationship between the attacker (the first parameter) and the attacked (the second parameter). After loading a Prolog source made up of these facts, the following semantics can be computed:

- admissible
- complete

[3]https:// www.dcc.fc.up.pt/ vsc/yap/

Figure 8.9: Main dialogue of Arguer.

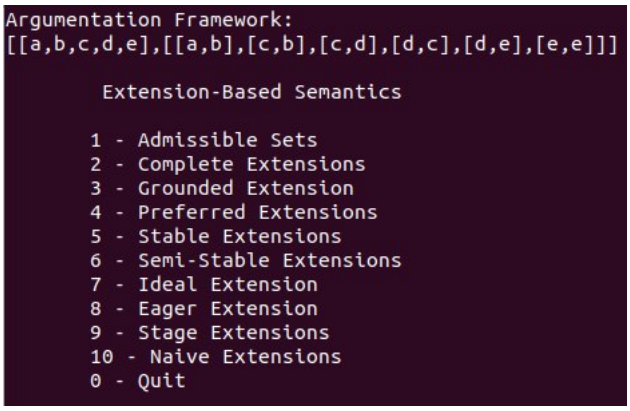


Figure 8.10: Main dialogue of Arguer.

- grounded
- preferred
- stable
- semi-stable
- eager
- stage
- naive.

Figure 8.10 shows the menu after loading an abstract AF knowledge source.

Figure 8.11 shows the admissible sets for the previously-loaded AF.

We report in Listing 8.6 the Prolog code for extracting admissible (and conflict-free) sets from arguments.

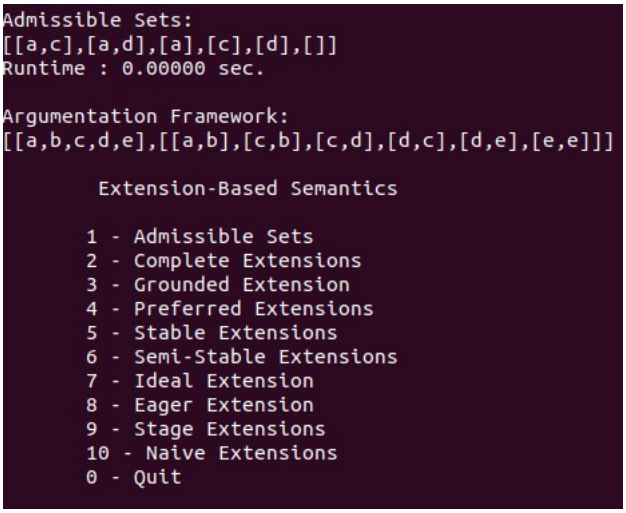


Figure 8.11: Main dialogue of Arguer.

Value-Based AF It is an extension of an abstract AF in which arguments are associated with values, and values play a crucial role, as well as attacks, in choosing acceptable sets of arguments. Apart from the predicates above, we can find `value/1` to express possible values and the `val_mapping/2` term to express that the argument in the first position is associated with the value in the second.
For this framework, the admissible, preferred and stable semantics can be computed.

Bipolar AF This extension provides the `support/2` term to express the opposite of the attack relationship.
For this framework, the admissible, preferred and stable semantics can be computed.

Weighted AF In this extension, for each argument, we can specify its authority in a domain. This is supported by the terms `domain/1` to specify domains and `authority/3` in which the first item is the argument, the second a value of authority, and the third the domain.

Bipolar Weighted AF In this extension, we have the same terms as the ones available in Bipolar AF, with the introduction of `rel_weight/3` expressing attacks or support based on the positivity of the weight on relationships. The range is $[-1, 0[\cup]0, 1]$.
For this framework, the strength-propagation ranking semantics can be computed.

Simplified Probabilistic AF In the proposed extension, the only added term is `likelihood/2` expressing the probability of each argument, and having range $]0, 1]$.

For this framework, the proposed max-probable semantics can be computed.

```
abs_arg(Argument) :-
    args:argument(Argument).

get_abstract_arguments(Arguments) :-
    findall(Argument, abs_arg(Argument), Arguments).

not_in_conflict([]) :-
    !.
not_in_conflict([Argument]) :-
    abs_arg(Argument),
    \+ attack_rel(Argument, Argument),
    !.

set_conflict_free_sets(Arguments) :-
    forall(subsets(Arguments, Args),
        (not_in_conflict(Args) ->
            assert(cf:cfree(Args)); true
        )
    ).

set_admissible_sets :-
    get_abstract_arguments(Arguments),
    set_conflict_free_sets(Arguments),
    get_conflict_free_sets(CFSets),
    set_acceptable_arguments(Arguments, CFSets).

get_conflict_free_sets(CFSets) :-
    findall(CFArgs, cf:cfree(CFArgs), CFSets).

set_acceptable_arguments(Arguments, CFSets) :-
    forall((member(CFSet, CFSets)),
```

```
32          (acceptable_set(Arguments, CFSet) ->
33              assertz(extensions:admissible(CFSet)); true
34          )
35      ).
```

Listing 8.6: Prolog Code for Conflict-Free Sets

8.2.5 Most-probable

Most-probable extension for a target argument is computed starting from the admissible sets, filtering those containing the target argument, and then an optimization function selects the arguments to be kept. Listing 8.7 shows the solution for probability updating after attacks, and how to normalize results for Answer Set Programming. ASP cannot manage floating numbers. For this reason, updated probabilities are normalized in the range $]0 - 100]$.

```
1  update(Arguments, Attacks, Likelihoods) :-
2      adjust_likelihoods(Arguments, Attacks, Likelihoods),
3      normalize_probabilities.
4
5  adjust_likelihoods([], _, _).
6  adjust_likelihoods([Arg | Rest], Attacks, Likelihoods) :-
7      adjust_likelihood(Arg, Attacks, Likelihoods),
8      adjust_likelihoods(Rest, Attacks, Likelihoods).
9
10 adjust_likelihood(Arg, Attacks, Likelihoods) :-
11     findall(X, lists:member([X,Arg], Attacks), Attackers),
12     findall(X, (lists:member(A, Attackers), lists:member([A
           ,X], Likelihoods)), ProbabilitiesAttackers), lists:
           member([Arg, OriginalProbability], Likelihoods),
13     Alfa is 0.2,
14     compute_product(Alfa, ProbabilitiesAttackers, Product),
15     AdjustedProbability is Product * OriginalProbability,
16     assertz(not_normalized_probability(Arg,
```

```
        AdjustedProbability)).

compute_product(Alfa, [], 1) :-
    !.

compute_product(Alfa, [P | Rest], Product) :-
    TermProduct is 1 - Alfa * P,
    compute_product(Alfa, Rest, OldProduct),
    Product is TermProduct * OldProduct.

find_max([X], X).
find_max([Number | Rest], Max) :-
    find_max(Rest, MaxRest),
    Max is max(Number, MaxRest).

normalize_probabilities :-
    findall(X, not_normalized_probability(_,X),
        Probabilities),
    find_max(Probabilities, Max),
    Factor is 100 / Max,
    findall((Arg, Probability), not_normalized_probability(
        Arg, Probability), ArgProbabilities),
    assert_probabilities(ArgProbabilities, Factor).

assert_probabilities([], Factor) :-
    !.
assert_probabilities([ArgProbability | Rest], Factor) :-
    ArgProbability = (Argument, Probability),
    NormalizedProbability is round(Factor * Probability),
    assertz(probability(Argument, NormalizedProbability)),
    assert_probabilities(Rest, Factor).
```

Listing 8.7: Prolog Code for Updating Likelihoods

Finally, Listing 8.8 shows the ASP code for computing most-prob extension. The listing shows the (normalized) example reported in Figure 8.8, and Listing 8.9 the result provided by ASP after optimization.

```
argument(a;b;c;d;e;f).
target(d).

attack(a, b).
attack(a, b).
attack(b, c).
attack(b, d).
attack(c, a).
attack(e, b).
attack(f, b).

likelihood(a, 80).
likelihood(b, 40).
likelihood(c, 80).
likelihood(d, 30).
likelihood(e, 70).
likelihood(f, 40).

% the target is always in the most-prob
most_prob(N) :- target(N).

%all the others are candidate
{most_prob(N) : argument(N)} :- not target(N).

%arguments must be defended (or not attacked) to be good
   candidates
:- most_prob(A), attack(B, A), not most_prob(C) : attack(C,
    B).

```

```
28 %take minimum likelihood
29 min_likelihood(Min) :- Min = #min { L : most_prob(A),
      likelihood(A, L), not target(A) }.
30
31 %take cardinality
32 size_most_prob(C) :- C = #count { A : most_prob(A) }.
33
34 %take set with the maximum minimum likelihood
35 #maximize { L : min_likelihood(L) }.
36
37 %add as many arguments as possible without affecting
      minimum likelihood
38 #maximize { C : size_most_prob(C) }.
39
40 #show most_prob/1.
```

Listing 8.8: ASP Code for Most-probable Extension

```
1 max_prob(d) max_prob(c) max_prob(e)
```

Listing 8.9: ASP Result for Most-probable Extension

The full implementation of Arguer and the Most-probable extension is available in

8.3 Ontology-based Instance Matching

In the context of ontology-driven data, instance matching is a relevant task in order to merge, correct and add information, as well as remove redundant information. The advantage of referring to schema-guided data is that you are aware of the possible values for some properties and a distance similarity metric can be defined for them.

8.3.1 Node Similarity

In this context, we define an ontology $\mathcal{O} = (\mathcal{C}, \mathcal{P}, \mathcal{R})$ where $\mathcal{C}$ is a set of classes, each of them uniquely identified by its name, $\mathcal{P}$ a named function $\mathcal{P} : \mathcal{C} \times \mathcal{D}$ from entities to a generic domain $\mathcal{D}$. Without losing generality, we suppose $\mathcal{D}$ can only be one of these

domains $\{Int, Categorical, Ordered, String\}$. Int represents a set of numbers within a range. $Ordered$ is a set of defined values like $Categorical$ but ordered like Likert scales. $\mathcal{P}$ expresses all the possible properties that can be defined for instances. If a property exists for a class at the ontological level, it means that it *might* be present also in instances. On the other hand, if a property is present for instances, it *must* exists also in the ontological level. $\mathcal{R} : \mathcal{C} \to \mathcal{C}$ is the set of relationships between entities. In this work, we do not take them into account. Among the relationships, *isA* is the relationship expressing the *subclass* ontological concept. This will be useful since instances can be compared only if they belong to the same entity or one is *subclass* of the other. We assume every instance belongs to exactly one class and that, for each class, there exists at most one *isA* relationship, i.e. single inheritance.

Given the above setting, we can now define how to measure differences between features of instances. Features are the peculiar characteristics of instances, described in the form of the above-mentioned properties. Remind that the domain $\mathcal{D}$ can be any of:
$\{Int, Categorical, Ordered, String\}$, and a distance metric must be defined for each. Specifically, we propose the following ones:

- **Int**: given $lower, upper \in \mathbb{N}, lower \leq upper$, a range $\mathcal{I} = [lower, upper]$ and $x, y \in \mathcal{I}$,

$$0 \leq \text{distance}(x, y) = \frac{2 \cdot \lceil \log_2 |\mathcal{I}| \rceil \cdot |x - y|}{|\mathcal{I}|} \leq 2 \cdot \lceil \log_2 |\mathcal{I}| \rceil |$$

- **Date**: given x, y two dates expressed as `yyyy/mm/dd`, distance(x, y) is equal to the ordinary number of days separating the two.

- **Categorical**: given a set of labels $\mathcal{L}$ and $x, y \in \mathcal{L}$,

$$\text{distance}(x, y) = \begin{cases} 1, & \text{if } x \neq y \\ 0, & \text{otherwise} \end{cases}$$

  Note that with $|\mathcal{L}| = 2$ (boolean case), we can map this case onto the Int with $\mathcal{I} = \{0, 1\}$ and we have distance$(0, 1) = \frac{2 \cdot \lceil \log_2 2 \rceil \cdot |0-1|}{2} = 1$ as expected.

- **Ordered**: given a set of labels $\mathcal{L}$ and a bijective function $\mathcal{O} : \mathcal{L} \to \{0, 1, ..., |L| -$

1} specifying the order and $x, y \in \mathcal{L}$,

$$0 \leq \text{distance}(x, y) = |O(x) - O(y)| \leq |\mathcal{L}|$$

- **String**: given an alphabet Σ and $x, y \in \Sigma^*$,

$$0 \leq \text{distance}(x, y) = DL(x, y) \leq max\{|x|, |y|\}$$

where $DL(x, y)$ is the Damerau-Levenshtein distance [292], particularly useful and suited for spell-checking.

In principle, an ontology designer may recognize which are the most prominent features to define similarity between instances of the same class. For instance, the features *name* and *surname* for **Person** will be much more useful than *age* or *title*. Suppose for each property $p \in \mathcal{P}$ we have a weight function $\mathcal{W} : \mathcal{P} \to \mathcal{R}$, the overall distance between two instances d_1, d_2 is

$$\text{distance}(d_1, d_2) = \frac{\sum_{p \in \mathcal{P}_{d1} \cap \mathcal{P}_{d2}} \text{distance}(p(d1), p(d2)) \cdot w(p)}{\sum_{p \in \mathcal{P}_{d1} \cap \mathcal{P}_{d2}} w(p)}$$

where $\mathcal{P}_d$ represents the set of properties available for instance d, and $p(d)$ the value of property p for instance d.

Unfortunately, it is not always the case for weighting properties for practical reasons given the time required in specifying all the weights, but also for the specificity of the domain knowledge required. For these reasons, it is much more convenient to identify a generic strategy based on types. As for the example about *name* and *surname*, it is intuitive to assume that *String* types are much more relevant in similarity computation. This is because the free text provides (in general) more specific information, and their equality/inequality provides quite often a good guess of how similar two instances are. If you reason with generic real-world instances you may think about people (often identified by name and surname), objects (which have name), places (which have names and addresses) and so on. For this reason, we prioritize similarities between strings over the others. Next, we claim that dates are stable, reflecting the date for which events happened. Apart from errors, it can be referred to as a meaningful feature for understanding similarity. Finally, integer values are less stable than categories, in the

sense that the number of times for which something happened is subject to changes over time. Following the above-mentioned idea of weighting, we now assign the same weight to all the features of the same type. In this case, it is much easier for an ontology designer to weigh only types, which should be a few in principle. Naming α_{string}, α_{date}, $\alpha_{categorical}$, $\alpha_{ordered}$, α_{int} the weights of types (respectively) $String$, $Date$, $Categorical$, $Ordered$, Int, and $\mathcal{T} = \{String, Date, Categorical, Ordered, Int\}$ the formula will be:

$$\text{distance}(d_1, d_2) = \frac{\sum_{t \in \mathcal{T}} \alpha_t \sum_{p \in \mathcal{P}_{d1} \cap \mathcal{P}_{d2}, \mathcal{P}(p)=t} \text{distance}(p(d1), p(d2))}{\sum_{t \in \mathcal{T}} \alpha_t \cdot |p \in \mathcal{P}_{d1} \cap \mathcal{P}_{d2}, \mathcal{P}(p) = t|}$$

Given the interpretable and ontology-based nature of the problem, we implemented the procedure for distance computation in Prolog. Listing 8.10 shows the main computation.

```
node_distance(N1, N2, AlfaInt, AlfaDate, AlfaCategorical,
    AlfaOrdered, AlfaString, Properties, Distance) :-
        findall(P1, property(N1, P1, _), Properties1),
        findall(P2, property(N2, P2, _), Properties2),
        findall(P, (member(P, Properties1), member(P,
            Properties2)), Properties),
        findall(P, (member(P, Properties), type(P, int)),
            IntProperties),
        findall(P, (member(P, Properties), type(P, date)),
            DateProperties),
        findall(P, (member(P, Properties), type(P,
            categorical)), CategoricalProperties),
        findall(P, (member(P, Properties), type(P, ordered)
            ), OrderedProperties),
        findall(P, (member(P, Properties), type(P, string))
            , StringProperties),
        node_int_distance(N1, N2, IntProperties, 0, DInt),
        node_date_distance(N1, N2, DateProperties, 0, DDate
            ),
        node_categorical_distance(N1, N2,
            CategoricalProperties, 0, DCategorical),
```

```
13          node_ordered_distance(N1, N2, OrderedProperties, 0,
                DOrdered),
14          node_string_distance(N1, N2, StringProperties, 0,
               DString),
15          Numerator is AlfaInt * DInt + AlfaDate * DDate +
               AlfaCategorical * DCategorical + AlfaOrdered *
               DOrdered + AlfaString * DString,
16          length(IntProperties, LInt),
17          length(DateProperties, LDate),
18          length(CategoricalProperties, LCategorical),
19          length(OrderedProperties, LOrdered),
20          length(StringProperties, LString),
21          Divisor is LInt * AlfaInt + LDate * AlfaDate +
               LCategorical * AlfaCategorical + LOrdered *
               AlfaOrdered + LString * AlfaString,
22          Distance is Numerator / Divisor.
23
24  entity(person). entity(student).
25  isA(student, person).
26  attribute(age, person).
27  attribute(date_of_birth, person).
28  attribute(is_worker, person).
29  attribute(qualification, person).
30  attribute(name, person).
31  type(age, int).
32  type(date_of_birth, date).
33  type(is_worker, categorical).
34  type(qualification, ordered).
35  type(name, string).
36  category(is_worker, 'yes').
37  category(is_worker, 'no').
38  order(qualification, 'high_school', 0).
```

```
39 order(qualification, 'bachelor', 1).
40 order(qualification, 'master', 2).
41 order(qualification, 'phd', 3).
42 node(1, person). node(2, student).
43 property(1, age, 52). property(2, age, 27).
44 property(1, date_of_birth, '1972/12/10'). property(2,
      date_of_birth, '1997/05/17').
45 property(1, is_worker, 'yes'). property(2, is_worker, 'no')
      .
46 property(1, qualification, 'phd'). property(2,
      qualification, 'master').
47 property(1, name, 'stefano'). property(2, name, 'davide').
```

Listing 8.10: Prolog Code for an example of distance computation

8.3.2 Experiments for Unfair Dataset Detection

In the realm of Machine Learning becoming more and more pervasive and crucial in our everyday life activities, the capability of understanding the process governing decision-making phases of algorithms is essential to connect to AI in a responsible and effective way. Basically, all the ML models learn by a first step of instance *training*, in which the model *learns* the correlation between input data and the target (correct in theory) label. The correlation between input and target will be the rationale for the algorithm to predict *unseen* data, the ones that we are interested in and make use of constantly.

In more recent years, the community of ML engineers, but not only, has raised the problem of how the models are trained. Specifically, they refer to the process of collecting and managing *training* data that the model is based on for the rest of its life. The problem we are mentioning is the dataset *unbalancement*, a sort of problem in which the dataset presents some *biases*, which can be defined as patterns that *should* not appear in the data because it does not reflect the reality of the domain we are describing. Many patterns emerging from datasets can be misleading, false, or appear only with the specificity of the data. Still, we are referring to those that can harm some categories of people, or that lead to *unfair* decision-making processes. Recognizing

unfair datasets or biases is even a recent line of research in which researchers endeavour to find techniques for detecting and (possibly solving) biases in data. One of the most frequent tasks for which a great extent of bias has been detected is the classification of criminals that, unfortunately, in many situations resulted in using ethnicity as one of the most distinguishing features [293].

Among the possible solutions to deal with this problem, much room has been dedicated to explainable or reasoning techniques. They are particularly suitable given their interpretable nature, and the capability of experts to evaluate the quality of the model. There are, on the other hand, techniques for improving ML models based on data perturbation or randomization. Perturbing is the process of (pseudo-)randomly varying values of instances to evaluate how the model behaves after perturbation. In the context of dataset evaluation, we exploit data perturbation to understand whether slight changes in the instance (most likely in the suspected biased features) lead to significant changes in the prediction. For instance, one of the basic questions we want to be able to answer is 'What happens if I change the ethnicity of the instance?'. Perturbating values can be done in many ways but, in order to create instances that make sense, we should know all the possible domains for the feature, so we know that 'caucasian' is a possible value to test while '42' is not. For this purpose, we work under the assumption that data is based on ontology (or schema). This is twofold: (i) we know exactly the domains of values for each property, and (ii) we have a strategy to measure instances of the same class. Without this assumption, an ontology alignment phase would be necessary.

8.3.3 First Results with Credit Cards Approval

A first analysis has been conducted on the Credit Approval dataset[4]. The dataset contains 690 instances and is composed of 15 features, of types Int, Real and Categorical. Some features regard gender and ethnicity, features that in principle should not be taken into account when deciding credit card approval The goal is to classify whether people At first, we performed a classification based on an interpretable model. Interpretability provided us with an easier way to determine which features to test first. We performed classification tasks with Decision Tree [294] and traditional split

[4]https://archive.ics.uci.edu/dataset/27/credit+approval

between train (80%) and test (20%).

Following decision rules, we detected that some relevant rules are governed by the "gender" or "ethnicity" of the person, features that should not be included in the classification process. For this reason, we generated instances distant **1** for the two features and reclassified all instances in the test set. Experiments showed that more than 5% of the people in the test set were classified differently only changing gender or ethnicity. Specifically, 3% of the population changed classification based on gender, and the 2% on ethnicity, and the intersection is empty.

The full implementation of the node similarity through ASP is available in the Appendix (A.2).

8.4 Discussion

In this chapter, I demonstrated how the integration of graphs with schemas can also be used with traditional and modern logic programming frameworks in order to solve some historic and modern problems with generic KGs.

Chapter 9

Final Remarks

9.1 Current Limitations

This work proposes to enrich the KRR field with a different perspective on how to model graph databases and exploit conceptualizations on them. However, some limitations or considerations need to be considered while working in this setting.

The main condition to be satisfied is the perfect consistency between data and schemas. This assumption, for obvious reasons, requires considerable effort in order to be preserved. Indeed, the GraphBRAIN technology offers strategies to preserve consistency; for instance, the online prototype has an interface that keeps you "safe", the API checks whether the operation is sound before actually performing it, and the proposed schema evaluation technique may also benefit the process of finding possible flaws in the conceptualization.

The mapping with the SW formalism requires manual labelling that, in most cases, requires an expert in the field. Yet for the purpose of keeping as much as possible the semantics of the GBS, I claim it is preferable in this context to propose a semi-automatic mapping, which was also historically considered valid by DB researchers.

Although offering higher performance than traditional databases[1] and libraries for data science tasks, full integration with ML libraries (e.g. written in Python) has not been developed yet, making other storage solutions more appropriate for full-fledged ML tasks.

[1] https://neo4j.com/news/how-much-faster-is-a-graph-database-really/

9.2 Future Work

This work may open scenarios for several lines of research. First of all, a representation including uncertainty or imprecise schemas can be considered. Moreover, following the RDF spirit, it may be explored the benefits of representing schemas in the graph itself, perhaps speeding up query times and understanding also whether traditional graph database tasks may benefit from this new representation.

Schemas can be enriched further, taking inspiration from the DL field, but also from rule-based programming. Understanding how to keep consistency while introducing rules in the schema is worth exploring as well.

Logic Programming (and its current extensions) and graphs still have much to be discovered. For instance, ML algorithms can benefit from graph constraint [295], and a full performance comparison is missing.

9.3 Conclusions

This thesis proposes a framework for graph database management and fruition, thanks to the introduction of graph schemas guiding the interaction. Throughout this work, I demonstrated several methodologies and uses to exploit schema-aware data. Given the similarity between graph structures, an approach for linking this framework to SW technologies is proposed. The mapping has been thought of as semi-automatic to link semantics to the original DB-based as well as logic programming solutions that, in correlation with schemas, perform better or open scenarios for new solutions. This work paves the way for several possibilities. First of all, in the context of SW some ad-hoc technology may try to reconcile graph DB instances with RDF instances. In the context of logical reasoning, it is easy to imagine other graph-based algorithms that can be improved with the introduction of schemas (e.g. Link Prediction among others). New schemas and domains of use emerge every day, giving rise to plenty of use cases and scenarios in which the approach can be reconsidered or extended. I claim this work paves the way for future researchers to explore strategies in which schemas gain advantage, perhaps also including certain degrees of uncertainty, which is inherently inevitable in the real world.

Bibliography

[1] Christof Strauch, Ultra-Large Scale Sites, and Walter Kriha. Nosql databases. *Lecture Notes, Stuttgart Media University*, 20(24):79, 2011.

[2] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A survey of link prediction in complex networks. *ACM computing surveys (CSUR)*, 49(4):1–33, 2016.

[3] Neil Zhenqiang Gong, Ameet Talwalkar, Lester Mackey, Ling Huang, Eui Chul Richard Shin, Emil Stefanov, Elaine Shi, and Dawn Song. Joint link prediction and attribute inference using a social-attribute network. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(2):1–20, 2014.

[4] Jie Chen, Yousef Saad, and Zechen Zhang. Graph coarsening: from scientific computing to machine learning. *SeMA Journal*, 79(1):187–223, 2022.

[5] Fan Chung. A brief survey of pagerank algorithms. *IEEE Trans. Netw. Sci. Eng.*, 1(1):38–42, 2014.

[6] Ronald R Yager. Fuzzy logic methods in recommender systems. *Fuzzy Sets and Systems*, 136(2):133–149, 2003.

[7] BA Ojokoh, MO Omisore, OW Samuel, and TO Ogunniyi. A fuzzy logic based personalized recommender system. *International Journal of Computer Science and Information Technology & Security (IJCSITS)*, 2(5):1008–1015, 2012.

[8] Peter Amey. Logic versus magic in critical systems. In *International Conference on Reliable Software Technologies*, pages 49–67. Springer, 2001.

[9] Nicola Guarino, Daniel Oberle, and Steffen Staab. What is an ontology? *Handbook on ontologies*, pages 1–17, 2009.

[10] B. Smith. Ontology. In *The furniture of the world*, pages 47–68. Brill, 2012.

[11] T. Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[12] Ronald Brachman and Hector Levesque. *Knowledge representation and reasoning*. Morgan Kaufmann, 2004.

[13] F. Stokman and P. Vries. Structuring knowledge in a graph. In *Human-computer interaction*, pages 186–206. Springer, 1988.

[14] Ivana Balazevic, Carl Allen, and Timothy Hospedales. Multi-relational poincaré graph embeddings. *Advances in Neural Information Processing Systems*, 32, 2019.

[15] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

[16] Maximilian Nickel and Volker Tresp. Tensor factorization for multi-relational learning. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III 13*, pages 617–621. Springer, 2013.

[17] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. Melo de, C. Gutierrez, S. Kirrane, J. Gayo, R. Navigli, S. Neumaier, et al. Knowledge graphs. *Synthesis Lectures on Data, Semantics, and Knowledge*, 12(2):1–257, 2021.

[18] S. Ji, S. Pan, E. Cambria, P. Marttinen, and S. Philip. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514, 2021.

[19] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610, 2014.

[20] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

[21] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[22] B. Abu-Salih. Domain-specific knowledge graphs: A survey. *Journal of Network and Computer Applications*, 185:103076, 2021.

[23] C. Feilmayr and W. Wöß. An analysis of ontologies and their success factors for application to business. *Data & Knowledge Engineering*, 101:1–23, 2016.

[24] Raymond Reiter. Nonmonotonic reasoning. In *Exploring artificial intelligence*, pages 439–481. Elsevier, 1988.

[25] Brian R Gaines. Foundations of fuzzy reasoning. *International Journal of Man-Machine Studies*, 8(6):623–668, 1976.

[26] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor. Industry-scale knowledge graphs: Lessons and challenges: Five diverse technology companies show how it's done. *Queue*, 17(2):48–75, 2019.

[27] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250, 2008.

[28] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195, 2015.

[29] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706, 2007.

[30] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. Hruschka, and T. Mitchell. Toward an architecture for never-ending language learning. In *Twenty-Fourth AAAI conference on artificial intelligence*, 2010.

[31] L. Heck, D. Hakkani-Tür, and G. Tur. Leveraging knowledge graphs for web-scale unsupervised semantic parsing. In *Proceedings of INTERSPEECH*, 2013.

[32] D. Damljanovic and K. Bontcheva. Named entity disambiguation using linked data. In *Proceedings of the 9th extended semantic web conference*, pages 231–240, 2012.

[33] Z. Zheng, X. Si, F. Li, E. Chang Y, and X. Zhu. Entity disambiguation with freebase. In *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, volume 1, pages 82–89. IEEE, 2012.

[34] R. Hoffmann, C. Zhang, X. Ling, L. Zettlemoyer, and D. Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, pages 541–550, 2011.

[35] J. Daiber, M. Jakob, C. Hokamp, and P. Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th international conference on semantic systems*, pages 121–124, 2013.

[36] A. Bordes, J. Weston, and N. Usunier. Open question answering with weakly supervised embedding models. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 165–180. Springer, 2014.

[37] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins. A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th annual Southeast regional conference*, pages 1–6, 2010.

[38] M. I. Jordan. *Learning in graphical models*, volume 89. Springer Science & Business Media, 1998.

[39] S. Batra and C. Tyagi. Comparative analysis of relational and graph databases. *International Journal of Soft Computing and Engineering (IJSCE)*, 2(2):509–512, 2012.

[40] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1–39, 2008.

[41] A. Silberschatz, H. Korth, and S. Sudarshan. Data models. *ACM Computing Surveys (CSUR)*, 28(1):105–108, 1996.

[42] E. Codd. Data models in database management. In *Proceedings of the 1980 workshop on Data abstraction, databases and conceptual modeling*, pages 112–114, 1980.

[43] J. Paredaens, P. Peelman, and L. Tanca. G-log: A graph-based query language. *IEEE Transactions on Knowledge and Data Engineering*, 7(3):436–453, 1995.

[44] S. Abiteboul. Querying semi-structured data. In *International Conference on Database Theory*, pages 1–18. Springer, 1997.

[45] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 505–516, 1996.

[46] R. Güting. Graphdb: Modeling and querying graphs in databases. In *VLDB*, volume 94, pages 12–15. Citeseer, 1994.

[47] I. Robinson, J. Webber, J. Webber, and E. Eifrem. *Graph Databases*. O'Reilly, 2013.

[48] J. Han, J. Pei, and H. Tong. *Data mining: concepts and techniques*. Morgan kaufmann, 2022.

[49] R. Kumar Kaliyar. Graph databases: A survey. In *International Conference on Computing, Communication & Automation*, pages 785–790. IEEE, 2015.

[50] Z. Huang, W. Chung, T. Ong, and H. Chen. A graph-based recommender system for digital library. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 65–73, 2002.

[51] J. Miller. Graph database applications and concepts with neo4j. In *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*, volume 2324, 2013.

[52] D. Anikin, O. Borisenko, and Y. Nedumov. Labeled property graphs: Sql or nosql? In *2019 Ivannikov Memorial Workshop (IVMEM)*, pages 7–13. IEEE, 2019.

[53] S. Raj. *Neo4j high performance*. Packt Publishing Ltd, 2015.

[54] D. Wang, W. Cui, and B. Qin. Ck-modes clustering algorithm based on node cohesion in labeled property graph. *Journal of Computer Science and Technology*, 34(5):1152–1166, 2019.

[55] C. T. Kalva, K. Abhishek, A. Vutnoori, and V. K. Maloth. Semantic filtering of twitter data using labeled property graph (lpg). *Journal of Computational and Theoretical Nanoscience*, 17(1):195–200, 2020.

[56] P. Formanowicz, M. Kasprzak, and P. Wawrzyniakr. *Labeled Graphs in Life Sciences—Two Important Applications*, pages 201–217. Springer International Publishing", address="Cham, 2022.

[57] E. Codd. A relational model of data for large shared data banks. In *Software pioneers*, pages 263–294. Springer, 2002.

[58] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[59] Achim Rettinger, Uta Lösch, Volker Tresp, Claudia d'Amato, and Nicola Fanizzi. Mining the semantic web. *Data Mining and Knowledge Discovery*, 24(3):613–662, 2012.

[60] B. Matthews. Semantic web technologies. *E-learning*, 6(6):8, 2005.

[61] P. Patel-Schneider. Owl web ontology language semantics and abstract syntax, w3c recommendation. *http://www. w3. org/TR/2004/REC-owl-semantics-20040210/*, 2004.

[62] Grigoris Antoniou and Frank van Harmelen. Web ontology language: Owl. *Handbook on ontologies*, pages 91–110, 2009.

[63] Y. Zou, T. Finin, and H. Chen. F-owl: An inference engine for semantic web. In *International Workshop on Formal Approaches to Agent-Based Systems*, pages 238–248. Springer, 2004.

[64] S. Decker, S. Melnik, F. Van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The semantic web: The roles of xml and rdf. *IEEE Internet computing*, 4(5):63–73, 2000.

[65] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):1–45, 2009.

[66] N. Baken. Linked data for smart homes: Comparing rdf and labeled property graphs. In *LDAC2020—8th Linked Data in Architecture and Construction Workshop*, pages 23–36, 2020.

[67] W. Xing and A. Ghorbani. Weighted pagerank algorithm. In *Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004.*, pages 305–314. IEEE, 2004.

[68] H. Deng, M. Lyu, and I. King. A generalized co-hits algorithm and its application to bipartite graphs. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 239–248, 2009.

[69] K. R. Apt. Logic programming. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, 1990:493–574, 1990.

[70] J. Svennevig. Abduction as a methodological approach to the study of spoken interaction, 2001.

[71] P. H. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995.

[72] Piero Bonatti, Francesco Calimeri, Nicola Leone, and Francesco Ricca. Answer set programming. *A 25-Year Perspective on Logic Programming: Achievements of the Italian Association for Logic Programming, GULP*, pages 159–182, 2010.

[73] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New generation computing*, 9:365–385, 1991.

[74] Leila Bayoudhi, Najla Sassi, and Wassim Jaziri. Towards a semantic querying approach for a multi-version owl 2 dl ontology. *International Journal of Computer Information Systems and Industrial Management Applications*, 13:11–11, 2021.

[75] Thomas E Malloy and Gerard L Hanley. Merlot: A faculty-focused web site of educational resources. *Behavior Research Methods, Instruments, & Computers*, 33(2):274–276, 2001.

[76] Amir Hossein Nabizadeh, José Paulo Leal, Hamed N Rafsanjani, and Rajiv Ratn Shah. Learning path personalization and recommendation methods: A survey of the state-of-the-art. *Expert Systems with Applications*, 159:113596, 2020.

[77] John K Tarus, Zhendong Niu, and Ghulam Mustafa. Knowledge-based recommendation: a review of ontology-based recommender systems for e-learning. *Artificial intelligence review*, 50:21–48, 2018.

[78] Oriana Licchelli, Teresa MA Basile, Nicola Di Mauro, Floriana Esposito, Giovanni Semeraro, and Stefano Ferilli. Machine learning approaches for inducing student models. In *Innovations in Applied Artificial Intelligence: 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2004, Ottawa, Canada, May 17-20, 2004. Proceedings 17*, pages 935–944. Springer, 2004.

[79] Dragan Gašević, Jelena Jovanović, and Vladan Devedžić. Ontology-based annotation of learning object content. *Interactive Learning Environments*, 15(1):1–26, 2007.

[80] Katrien Verbert and Erik Duval. Alocom: a generic content model for learning objects. *International Journal on Digital Libraries*, 9:41–63, 2008.

[81] Dimitrios A Koutsomitropoulos, Andreas D Andriopoulos, and Spiridon D Likothanassis. Semantic classification and indexing of open educational resources with word embeddings and ontologies. *Cybernetics and Information Technologies*, 20(5):95–116, 2020.

[82] Dimitrios A Koutsomitropoulos and Georgia D Solomou. A learning object ontology repository to support annotation and discovery of educational resources using semantic thesauri. *IFLA journal*, 44(1):4–22, 2018.

[83] Saeed Rouhani and Seyed Vahid Mirhosseini. Development and evaluation of intelligent agent-based teaching assistant in e-learning portals. *International Journal of Web-Based Learning and Teaching Technologies (IJWLTT)*, 10(4):52–62, 2015.

[84] Vlad Fernoagă, George-Alex Stelea, Cristinel Gavrilă, and Florin Sandu. Intelligent education assistant powered by chatbots. *eLearning & Software for Education*, 2, 2018.

[85] Harm Pinkster. Sintassi e semantica latina. *(No Title)*, 1991.

[86] Charlton Thomas Lewis and Hugh Macmaster Kingery. *An elementary Latin dictionary*. American Book Company, 1915.

[87] Marianne Pade. *LATIN AND THE EARLY MODERN WORLD: linguistic identity and the polity from*. 2016.

[88] Witold Manczak. József herman, vulgar latin, translated by roger wright, the pennsylvania state university press, university park, pennsylvania, xiv+ 130 p. *Linguistica*, 41(1):163–166, 2001.

[89] James Clackson. *A companion to the Latin language*. John Wiley & Sons, 2011.

[90] Jürgen Leonhardt. *Latin: Story of a world language*. Harvard University Press, 2013.

[91] Philipp Roelli. *Latin as the Language of Science and Learning*. de Gruyter, 2021.

[92] James Noel Adams, Mark Janse, and Simon Swain. *Bilingualism in ancient society: Language contact and the written text*. Oxford University Press, USA, 2002.

[93] Kathryn Allan, Justyna A Robinson, et al. *Current methods in historical semantics*. De Gruyter Mouton Berlin, 2012.

[94] Marco Passarotti, Francesco Mambrini, Greta Franzini, Flavio Massimiliano Cecchini, Eleonora Litta, Giovanni Moretti, Paolo Ruffolo, and Rachele Sprugnoli. Interlinking through lemmas. the lexical collection of the lila knowledge base of linguistic resources for latin. *Studi e Saggi Linguistici*, 58(1):177–212, 2020.

[95] Anas Fahad Khan, Christian Chiarcos, Thierry Declerck, Daniela Gifu, Elena González-Blanco García, Jorge Gracia, Maxim Ionov, Penny Labropoulou, Francesco Mambrini, John P McCrae, et al. When linguistics meets web technologies. recent advances in modelling linguistic linked data. *Semantic Web*, 13(6):987–1050, 2022.

[96] Florentina Armaselu, Elena-Simona Apostol, Anas Fahad Khan, Chaya Liebeskind, Barbara McGillivray, Ciprian-Octavian Truică, Andrius Utka, Giedrė Valūnaitė Oleškevičienė, and Marieke van Erp. Ll (o) d and nlp perspectives on semantic change for humanities research. *Semantic Web*, 13(6):1051–1080, 2022.

[97] Fahad Khan, John Philip McCrae, Francisco Javier Minaya Gómez, Rafael Cruz González, and Javier E Díaz-Vera. Some considerations in the construction of a historical language wordnet. In *Proceedings of the 12th Global Wordnet Conference*, pages 101–105, 2023.

[98] Anas Fahad Khan. Towards the representation of etymological data on the semantic web. *Information*, 9(12):304, 2018.

[99] Fahad Khan. Representing temporal information in lexical linked data resources. In *Proceedings of the 7th Workshop on Linked Data in Linguistics (LDL-2020)*, pages 15–22, 2020.

[100] Barbara McGillivray and Gard B Jenset. Quantifying the quantitative (re-) turn in historical linguistics. *Humanities and Social Sciences Communications*, 10(1):1–6, 2023.

[101] Pierpaolo Basile, Pierluigi Cassotti, Stefano Ferilli, Barbara McGillivray, et al. A new time-sensitive model of linguistic knowledge for graph databases. In *AI4CH@ AI* IA*, pages 69–80, 2022.

[102] Stefano Minozzi. Latin wordnet, una rete di conoscenza semantica per il latino e alcune ipotesi di utilizzo nel campo dell'information retrieval. *Strumenti digitali e collaborativi per le Scienze dell'Antichita*, 14:123–134, 2017.

[103] Erica Biagetti, Chiara Zanchi, and William Michael Short. Toward the creation of wordnets for ancient indo-european languages. In *Proceedings of the 11th Global Wordnet Conference*, pages 258–266, 2021.

[104] Dominik Schlechtweg, Barbara McGillivray, Simon Hengchen, Haim Dubossarsky, and Nina Tahmasebi. Semeval-2020 task 1: Unsupervised lexical semantic change detection. *arXiv preprint arXiv:2007.11464*, 2020.

[105] Barbara McGillivray. Latin lexical semantic annotation. *Oxford Text Archive Core Collection*, 2021.

[106] Barbara McGillivray, Daria Kondakova, Annie Burman, Francesca Dell'Oro, Helena Bermúdez Sabel, Paola Marongiu, and Manuel Márquez Cruz. A new corpus annotation framework for latin diachronic lexical semantics. *Journal of Latin Linguistics*, 21(1):47–105, 2022.

[107] Nidhi Rajesh Mavani, Jarinah Mohd Ali, Suhaili Othman, MA Hussain, Haslaniza Hashim, and Norliza Abd Rahman. Application of artificial intelligence in food industry—a guideline. *Food Engineering Reviews*, 14(1):134–175, 2022.

[108] Chakkrit Snae and Michael Bruckner. Foods: a food-oriented ontology-driven system. In *2008 2nd ieee international conference on digital ecosystems and technologies*, pages 168–176. IEEE, 2008.

[109] Raciel Yera Toledo, Ahmad A Alzahrani, and Luis Martinez. A food recommender system considering nutritional information and user preferences. *IEEE Access*, 7:96695–96711, 2019.

[110] Gorjan Popovski, Barbara Korousic-Seljak, and Tome Eftimov. Foodontomap: Linking food concepts across different food ontologies. In *KEOD*, pages 195–202, 2019.

[111] Duygu Çelik et al. Foodwiki: Ontology-driven mobile safe food consumption system. *The scientific World journal*, 2015, 2015.

[112] Nur Aini Rakhmawati, Jauhar Fatawi, Ahmad Choirun Najib, and Azmi Adi Firmansyah. Linked open data for halal food products. *Journal of King Saud University-Computer and Information Sciences*, 33(6):728–739, 2021.

[113] Maged N Kamel Boulos, Abdulslam Yassine, Shervin Shirmohammadi, Chakkrit Snae Namahoot, and Michael Brückner. Towards an "internet of food": food ontologies for the internet of things. *Future Internet*, 7(4):372–392, 2015.

[114] Li Qin, Zhigang Hao, and Liang Zhao. Food safety knowledge graph and question answering system. In *Proceedings of the 2019 7th International Conference on Information Technology: IoT and Smart City*, pages 559–564, 2019.

[115] Vasvi Bajaj, Rajat Bhusan Panda, Chetna Dabas, and Parmeet Kaur. Graph database for recipe recommendations. In *2018 7th international conference on reliability, infocom technologies and optimization (Trends and future directions)(ICRITO)*, pages 1–6. IEEE, 2018.

[116] Pat Riva, Patrick Le Bœuf, and Maja Žumer. Ifla library reference model. *A Conceptual Model for Bibliographic Information. Hg. v. IFLA International Federation of Library Associations and institutions. Online verfügbar unter https://www. ifla. org/files/assets/cataloguing/frbr-lrm/ifla-lrm-august-2017_ rev201712. pdf*, 2017.

[117] Maja Žumer. Ifla library reference model (ifla lrm)—harmonisation of the frbr family. *KO Knowledge Organization*, 45(4):310–318, 2018.

[118] Annika Hinze, George Buchanan, David Bainbridge, and Ian H Witten. Greenstone: a platform for semantic digital libraries, 2007.

[119] Getaneh Alemu, Brett Stevens, Penny Ross, and Jane Chandler. Linked data for libraries: benefits of a conceptual shift from library-specific record structures to rdf-based data models. *New library world*, 113(11/12):549–570, 2012.

[120] Joffrey Decourselle. Case-oriented semantic enrichment of bibliographic entities. In *Theory and Practice of Digital Libraries*, 2016.

[121] Joffrey Decourselle. Towards a pattern-based semantic enrichment of bibliographic entities. *IEEE TCDL*, 12(2), 2016.

[122] Preedip Balaji Babu, Amit K Sarangi, and Devika P Madalli. Knowledge organization systems for semantic digital libraries. 2012.

[123] Sebastian Ryszard Kruk, Bernhard Haslhofer, Piotr Piotrowski, Adam Westerski, and Tomasz Woroniecki. The role of ontologies in semantic digital libraries. In *European Networked Knowledge Organization Systems (NKOS) Workshop*. Alicante Spain, 2006.

[124] Carlo Bianchini. The entities of the ifla-lrm, ric-cm and cidoc-crm models in the semantic web. *JLIS. it*, 13(3):63–75, 2022.

[125] María Hallo, Sergio Luján-Mora, Alejandro Maté, and Juan Trujillo. Current state of linked data in digital libraries. *Journal of Information Science*, 42(2):117–127, 2016.

[126] Alexandre Passant and Philippe Laublet. Meaning of a tag: A collaborative approach to bridge the gap between tagging and linked data. *LDOW*, 369, 2008.

[127] Sebastian Ryszard Kruk, Marcin Synak, and Kerstin Zimmermann. Marcont–integration ontology for bibliographic description formats. In *Proceedings of the International Conference on Dublin Core and Metadata Applications*. Dublin Core Metadata Initiative, 2005.

[128] Javier Lacasta, Javier Nogueras-Iso, Gilles Falquet, Jacques Teller, and F Javier Zarazaga-Soria. Design and evaluation of a semantic enrichment process for bibliographic databases. *Data & Knowledge Engineering*, 88:94–107, 2013.

[129] Stefano Borgo, Roberta Ferrario, Aldo Gangemi, Nicola Guarino, Claudio Masolo, Daniele Porello, Emilio M Sanfilippo, and Laure Vieu. Dolce: A descriptive ontology for linguistic and cognitive engineering. *Applied ontology*, 17(1):45–69, 2022.

[130] Helena Simões Patrício, Maria Inês Cordeiro, and Pedro Nogueira Ramos. Formalizing enrichment mechanisms for bibliographic ontologies in the semantic web. In *Research Conference on Metadata and Semantics Research*, pages 147–158. Springer, 2018.

[131] Said Fathalla, Sahar Vahdati, Sören Auer, and Christoph Lange. Semsur: a core ontology for the semantic representation of research findings. *Procedia Computer Science*, 137:151–162, 2018.

[132] Susmita Sadhu, Poonam Anthony, Plaban Kumar Bhowmick, and Debarshi Kumar Sanyal. Towards development of ontology for the national digital library of india. *Available on-line at https://savesd. github. io/2018/submission/sadhu/index. pdf (consulted 6 September 2023)*.

[133] L Jael García Castro, Olga X Giraldo, and Alexander García Castro. Using the annotation ontology in semantic digital libraries. In *Proceedings of the 2010 International Conference on Posters & Demonstrations Track*, volume 658, pages 153–156. Citeseer, 2010.

[134] Antonio Martín Montes and Carlos León de Mora. Expert knowledge management based on ontology in a digital library. In *12th International Conference on Enterprise Information Systems: Funchal, Madeira (Portugal), 8-12 de junio, 2010, 291-298*, 2010.

[135] W Bruce Croft and Roger H Thompson. I3r: A new approach to the design of document retrieval systems. *Journal of the american society for information science*, 38(6):389–404, 1987.

[136] Helmut Berger, Michael Dittenbach, and Dieter Merkl. An adaptive information retrieval system based on associative networks. In *Proceedings of the first Asian-Pacific conference on Conceptual modelling-Volume 31*, pages 27–36, 2004.

[137] Richard K Belew. Adaptive information retrieval: using a connectionist representation to retrieve and leasrn about documents. In *ACM SIGIR Forum*, volume 51, pages 106–115. ACM New York, NY, USA, 2017.

[138] Edward A Fox. Development of the coder system: A testbed for artificial intelligence methods in information retrieval. *Information Processing & Management*, 23(4):341–366, 1987.

[139] Edward A Fox and Robert K France. Architecture of an expert system for composite document analysis, representation, and retrieval. *International Journal of Approximate Reasoning*, 1(2):151–175, 1987.

[140] Marcos André Gonçalves, Robert K France, Edward A Fox, and Tamas E Doszkocs. Marian searching and querying across heterogeneous federated digital libraries. In *DELOS*, 2000.

[141] Eugen Popovici. *Information Retrieval of Text, Structure and Sequential Data in Heterogeneous XML Document Collections.* PhD thesis, Université de Bretagne Sud; Université Européenne de Bretagne, 2008.

[142] R. Meersman. Ontologies and databases: More than a fleeting resemblance. *STAR*, 3, 2001.

[143] H. Bulskov, R. Knappe, and T. Andreasen. On querying ontologies and databases. In *International Conference on Flexible Query Answering Systems*, pages 191–202. Springer, 2004.

[144] D. Calvanese, G. Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, and R. Rosati. Ontologies and databases: The dl-lite approach. In *Reasoning Web International Summer School*, pages 255–356. Springer, 2009.

[145] S. Ranu and A. K. Singh. Graphsig: A scalable approach to mining significant subgraphs in large graph databases. In *2009 IEEE 25th International Conference on Data Engineering*, pages 844–855. IEEE, 2009.

[146] C. Wang, W. Wang, J. Pei, Y. Zhu, and B. Shi. Scalable mining of large disk-based graph databases. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 316–325, 2004.

[147] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 721–724. IEEE, 2002.

[148] M. Hegland. The apriori algorithm–a tutorial. *Mathematics and computation in imaging science and information processing*, pages 209–262, 2007.

[149] J. Huan, W. Wang, J. Prins, and J. Yang. Spin: mining maximal frequent subgraphs from graph databases. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–586, 2004.

[150] R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

[151] W. D. Wilson and T. R. Martinez. Instance pruning techniques. In *ICML*, volume 97, pages 400–411, 1997.

[152] S. U. Rehman, Au. U. Khan, and S. Fong. Graph mining: A survey of graph mining techniques. In *Seventh International Conference on Digital Information Management (ICDIM 2012)*, pages 88–92, 2012.

[153] S. Ceri, G. Gottlob, and L. Tanca. *Logic programming and databases*. Springer Science & Business Media, 2012.

[154] D. G. Bridge. Towards conversational recommender systems: A dialogue grammar approach. In *ECCBR workshops*, pages 9–22, 2002.

[155] A. Markus, Z. Márkusz, J. Farkas, and J. Filemon. Fixture design using prolog: an expert system. *Robotics and computer-integrated manufacturing*, 1(2):167–172, 1984.

[156] M. Das, Y. Wu, T. Khot, K. Kersting, and S. Natarajan. Scaling lifted probabilistic inference and learning via graph databases. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 738–746. SIAM, 2016.

[157] K. Stokes. Cover-up: a probabilistic privacy-preserving graph database model. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–8, 2019.

[158] N. Zaki, T. Chandana, and A. A. Hany. Knowledge graph construction and search for biological databases. In *2017 International Conference on Research and Innovation in Information Systems (ICRIIS)*, pages 1–6, 2017.

[159] V.S. Silva, A. Freitas, and S. Handschuh. Building a knowledge graph from natural language definitions for interpretable text entailment recognition. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. European Language Resources Association (ELRA), may 2018.

[160] C. Fellbaum. Wordnet. In *Theory and applications of ontology: computer applications*, pages 231–243. Springer, 2010.

[161] I. Dagan, O. Glickman, and B. Magnini. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop*, pages 177–190. Springer, 2005.

[162] L. Penev, M. Dimitrova, V. Senderov, G. Zhelezov, T. Georgiev, P. Stoev, and K. Simov. Openbiodiv: A knowledge graph for literature-extracted linked open data in biodiversity science. *Pensoft Publishers*, 7(2), 2019.

[163] F. Bauer and M. Kaltenböck. Linked open data: The essentials. *Edition mono/-monochrom, Vienna*, 710, 2011.

[164] S. Purohit, N. Van, and George Chin. Semantic property graph for scalable knowledge graph analytics. *arXit*, Sep 2020.

[165] N. Mitrou D.E. Spanos, P. Stavrou. Bringing relational databases into the semantic web: A survey. *Semantic Web*, 3:169–209, 2012.

[166] K.N. Vavliakis, T.K. Grollios, and P.A. Mitkas. Rdote–publishing relational databases into the semantic web. *Journal of Systems and Software*, 86(1):89–99, 2013.

[167] R. Angles and C. Gutierrez. Querying rdf data from a graph database perspective. In *The Semantic Web: Research and Applications*, pages 346–360. Springer International Publishing, 2005.

[168] S. Sakr, S. Elnikety, and Y. He. G-sparql: A hybrid engine for querying large attributed graphs. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 335—-344. Association for Computing Machinery, 2012.

[169] L. Libkin, J. Reutter, and D. Vrgoč. Trial for rdf: adapting graph query languages for rdf data. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 201–212, 2013.

[170] H. Thakkar, D. Punkani, Y. Keswani, J. Lehmann, and S. Auer. A stitch in time saves nine–sparql querying of property graphs using gremlin traversals. *arXiv preprint arXiv:1801.02911*, 2018.

[171] M.A Rodriguez. The gremlin graph traversal machine and language (invited talk). In *Proceedings of the 15th Symposium on Database Programming Languages*, pages 1–10, 2015.

[172] R. De Virgilio. Smart rdf data storage in graph databases. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 872–881. IEEE, 2017.

[173] D. Tomaszuk. Rdf data in property graph model. In *Research Conference on Metadata and Semantics Research*, pages 104–115. Springer, 2016.

[174] B. Iordanov. Hypergraphdb: a generalized graph database. In *International conference on web-age information management*, pages 25–36. Springer, 2010.

[175] S. Das, M. Perry, J. Srinivasan, and E. Chong. A tale of two graphs: Property graphs as rdf in oracle. In *EDBT*, pages 762–773, 2014.

[176] H. Chiba, R. Yamanaka, Y. Keswani, and S. Matsumoto. G2gml: Graph to graph mapping language for bridging rdf and property graphs. In *International Semantic Web Conference*, pages 160–175. Springer, 2020.

[177] S. Matsumoto, R. Yamanaka, and H. Chiba. Mapping rdf graphs to property graphs. *arXiv preprint arXiv:1812.01801*, 2018.

[178] R. Angles, H. Thakkar, and D. Tomaszuk. Mapping rdf databases to property graph databases. *IEEE Access*, 8:86091–86110, 2020.

[179] D. Tomaszuk, R. Angles, and H. Thakkar. Pgo: Describing property graphs in rdf. *IEEE Access*, 8:118355–118369, 2020.

[180] Alexander Schätzle, Martin Przyjaciel-Zablocki, Thorsten Berberich, and Georg Lausen. S2x: graph-parallel querying of rdf with graphx. In *Biomedical data management and graph online querying*, pages 155–168. Springer, 2015.

[181] E. Haihong, P. Han, and M. Song. Transforming rdf to property graph in hugegraph. In *Proceedings of the 6th International Conference on Engineering and MIS 2020*, ICEMIS'20. Association for Computing Machinery, 2020.

[182] R. Zhang, P. Liu, X. Guo, S. Li, and X. Wang. A unified relational storage scheme for rdf and property graphs. In *International Conference on Web Information Systems and Applications*, pages 418–429. Springer, 2019.

[183] R. Angles, H. Thakkar, and D. Tomaszuk. Rdf and property graphs interoperability: Status and issues. In *AMW*, 2019.

[184] P. Pareti and G. Konstantinidis. A review of shacl: From data validation to schema reasoning for rdf graphs. *Reasoning Web International Summer School*, pages 115–144, 2021.

[185] S. Staworko, I. Boneva, J. E. L. Gayo, S. Hym, E. G. Prud'Hommeaux, and H. Solbrig. Complexity and expressiveness of shex for rdf. In *18th International Conference on Database Theory (ICDT 2015)*, 2015.

[186] U. Brandes, M. Eiglsperger, J. Lerner, and C. Pich. Graph markup language (graphml), 2013.

[187] Dotml. `http://martin-loetzsch.de/DOTML/`. Accessed: 2022-08-20.

[188] G. G. V. Yon. Building, importing, and exporting gexf graph files with rgexf. *Journal of Open Source Software*, 6(64):3456, 2021.

[189] Graphson. `https://docs.oracle.com/en/database/oracle/property-graph/21.1/spgdg/graphson-data-format.html`. Accessed: 2022-08-20.

[190] V. Nguyen, H. Y. Yip, H. Thakkar, Q. Li, E. Bolton, and O. Bodenreider. Singleton property graph: Adding a semantic web abstraction layer to graph databases. In *BlockSW/CKG@ ISWC*, 2019.

[191] O. Hartig. Reconciliation of rdf* and property graphs. *arXiv preprint arXiv:1409.3288*, 2014.

[192] G. Modoni, M. Sacco, and W. Terkaj. A survey of rdf store solutions. In *2014 International Conference on Engineering, Technology and Innovation (ICE)*, pages 1–7. IEEE, 2014.

[193] Mariano Fernández-López. Overview of methodologies for building ontologies. 1999.

[194] Cheng-Tao Chu, Sang Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Kunle Olukotun, and Andrew Ng. Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19, 2006.

[195] Hanâ Lbath, Angela Bonifati, and Russ Harmer. Schema inference for property graphs. In *EDBT 2021-24th International Conference on Extending Database Technology*, pages 499–504, 2021.

[196] Ahmed E Samy, Lodovico Giaretta, Zekarias T Kefato, and Šarūnas Girdzijauskas. Schemawalk: Schema aware random walks for heterogeneous graph embedding. In *Companion Proceedings of the Web Conference 2022*, pages 1157–1166, 2022.

[197] Grigoris Antoniou and Athanasios Kehagias. A note on the refinement of ontologies. *International Journal of Intelligent Systems*, 15(7):623–632, 2000.

[198] Monica Palmirani, Giorgia Bincoletto, Valentina Leone, Salvatore Sapienza, Francesco Sovrano, et al. Pronto ontology refinement through open knowledge extraction. *FRONTIERS IN ARTIFICIAL INTELLIGENCE AND APPLICATIONS*, 322:205–210, 2019.

[199] Enrique Alfonseca and Suresh Manandhar. Improving an ontology refinement method with hyponymy patterns. *cell*, 4081:0–0087, 2002.

[200] Enrique Alfonseca and Suresh Manandhar. Proposal for evaluating ontology refinement methods. In *LREC*. Citeseer, 2002.

[201] Dionysios D Kehagias, Ioannis Papadimitriou, Joana Hois, Dimitrios Tzovaras, and John Bateman. A methodological approach for ontology evaluation and refinement. In *ASK-IT Final Conference. June.(Cit. on p.)*, pages 1–13, 2008.

[202] Mamoru Ohta, Kouji Kozaki, and Riichiro Mizoguchi. A quality assurance framework for ontology construction and refinement. In *Advances in Intelligent Web Mastering–3: Proceedings of the 7th Atlantic Web Intelligence Conference, AWIC 2011, Fribourg, Switzerland, January, 2011*, pages 207–216. Springer, 2011.

[203] Sarika Jain and Valerie Meyer. Evaluation and refinement of emergency situation ontology. *International Journal of Information and Education Technology*, 8(10):713–719, 2018.

[204] Angela Bonifati, Stefania Dumbrava, and Nicolas Mir. Hierarchical clustering for property graph schema discovery. In *EDBT 2022: 25th International Conference on Extending Database Technology*, pages 449–453. OpenProceedings. org, 2022.

[205] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, et al. Pg-schema: Schemas for property graphs. *Proceedings of the ACM on Management of Data*, 1(2):1–25, 2023.

[206] Ansgar Scherp, David Richerby, Till Blume, Michael Cochez, and Jannik Rau. Structural Summarization of Semantic Graphs Using Quotients. *Transactions on Graph Data and Knowledge*, 1(1):12:1–12:25, 2023.

[207] Olaf Hartig and Jorge Pérez. Semantics and complexity of graphql. In *Proceedings of the 2018 World Wide Web Conference*, pages 1155–1164, 2018.

[208] Mónica Figuera, Philipp D Rohde, and Maria-Esther Vidal. Trav-shacl: Efficiently validating networks of shacl constraints. In *Proceedings of the Web Conference 2021*, pages 3337–3348, 2021.

[209] Wojciech Czerwiński, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for petri nets is not elementary. *Journal of the ACM (JACM)*, 68(1):1–28, 2020.

[210] Chao Zhang, Angela Bonifati, and M Tamer Özsu. An overview of reachability indexes on graphs. In *Companion of the 2023 International Conference on Management of Data*, pages 61–68, 2023.

[211] Huibin Wang, Ming Chen, and Xianglin Wei. A k-hop constrained reachability based proactive connectivity maintaining mechanism of uav swarm networks. *Journal of Internet Technology*, 24(6):1329–1341, 2023.

[212] Tobia Marcucci, Jack Umenberger, Pablo Parrilo, and Russ Tedrake. Shortest paths in graphs of convex sets. *SIAM Journal on Optimization*, 34(1):507–532, 2024.

[213] Arend Rensink. Representing first-order logic using graphs. In *International Conference on Graph Transformation*, pages 319–335. Springer, 2004.

[214] Matteo Acclavio, Ross Horne, and Lutz Straßburger. Logic beyond formulas: A proof system on graphs. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 38–52, 2020.

[215] Keijo Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe petri nets. *Fundamenta Informaticae*, 37(3):247–268, 1999.

[216] Flavio Cruz, Ricardo Rocha, Seth Copen Goldstein, and Frank Pfenning. A linear logic programming language for concurrent programming over graph structures. *Theory and Practice of Logic Programming*, 14(4-5):493–507, 2014.

[217] Despoina Magka, Boris Motik, and Ian Horrocks. Modelling structured domains using description graphs and logic programming. In *Extended Semantic Web Conference*, pages 330–344. Springer, 2012.

[218] Rose Catherine and William Cohen. Personalized recommendations using knowledge graphs: A probabilistic logic programming approach. In *Proceedings of the 10th ACM conference on recommender systems*, pages 325–332, 2016.

[219] Nada Lavrac and Saso Dzeroski. Inductive logic programming. In *WLP*, pages 146–160. Springer, 1994.

[220] Domen Smole, Marjan Čeh, and Tomaž Podobnikar. Evaluation of inductive logic programming for information extraction from natural language texts to support spatial data recommendation services. *International Journal of Geographical Information Science*, 25(11):1809–1827, 2011.

[221] Fernando Orejas, Hartmut Ehrig, and Ulrike Prange. A logic of graph constraints. In *International Conference on Fundamental Approaches to Software Engineering*, pages 179–198. Springer, 2008.

[222] S. Ferilli and D. Redavid. The graphbrain system for knowledge graph management and advanced fruition. In *International Symposium on Methodologies for Intelligent Systems*, pages 308–317. Springer, 2020.

[223] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[224] Melvil Dewey. *A classification and subject index, for cataloguing and arranging the books and pamphlets of a library*. Brick row book shop, Incorporated, 1876.

[225] S. Ferilli. Integration strategy and tool between formal ontology and graph database technology. *MDPI*, Oct 2021.

[226] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM transactions on database systems (TODS)*, 1(1):9–36, 1976.

[227] Bernhard Thalheim. Extended entity-relationship model. *Encyclopedia of database systems*, 1:1083–1091, 2009.

[228] Ramez Elmasri and Shamkant B Navathe. Fundamentals of database systems seventh edition, 2016.

[229] Terry Halpin. *Object-role modeling fundamentals: a practical guide to data modeling with ORM*. Technics Publications, 2015.

[230] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2018.

[231] Wolfgang Nejdl, Martin Wolpers, Christian Capelle, Rechnergest Wissensverarbeitung, et al. The rdf schema specification revisited. *In Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, Modellierung 2000*, 2000.

[232] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F Patel-Schneider, Sebastian Rudolph, et al. Owl 2 web ontology language primer. *W3C recommendation*, 27(1):123, 2009.

[233] Mary Brady, Carmelo Montanez-Rivera, Richard Rivello, and Lisa Carnahan. Understanding the xml (extensible markup). 2001.

[234] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of json schema. In *Proceedings of the 25th international conference on World Wide Web*, pages 263–273, 2016.

[235] Namespace-based Validation Dispatching. Schema definition languages (dsdl)—. 2006.

[236] Shudi Sandy Gao, C Michael Sperberg-McQueen, and Henry Thompson. W3c xml schema definition language (xsd) 1.1 part 1: Structures. 2012.

[237] Angela Bonifati, Peter Furniss, Alastair Green, Russ Harmer, Eugenia Oshurko, and Hannes Voigt. Schema validation and evolution for graph databases. In *Conceptual Modeling: 38th International Conference, ER 2019, Salvador, Brazil, November 4–7, 2019, Proceedings 38*, pages 448–456. Springer, 2019.

[238] Jim Melton. Database language sql. In *Handbook on Architectures of Information Systems*, pages 103–128. Springer, 2006.

[239] Anthony Papantonakis and Peter JH King. Gql, a declarative graphical query language based on the functional data model. In *Proceedings of the workshop on Advanced visual interfaces*, pages 113–122, 1994.

[240] Sandro Bimonte, Enrico Gallinucci, Patrick Marcel, and Stefano Rizzi. Data variety, come as you are in multi-model data warehouses. *Information Systems*, 104:101734, 2022.

[241] Norbert Zaniewicz and Andrzej Salamończyk. Comparison of mongodb, neo4j and arangodb databases using the developed data generator for nosql databases. *Studia Informatica. System and information technology*, 26(1):61–72, 2022.

[242] Behrad Babaee, Damon Daylamani-Zad, and Ken Tune. Co 2 emission efficiency as a measurable non-functional requirement: An emission estimation framework. *IEEE Access*, 10:97576–97585, 2022.

[243] Jéssica Monteiro, Filipe Sá, and Jorge Bernardino. Graph databases assessment: Janusgraph, neo4j, and tigergraph. In *Perspectives and Trends in Education and Technology: Selected Papers from ICITED 2022*, pages 655–665. Springer, 2023.

[244] Min Wu, Xinglu Yi, Hui Yu, Yu Liu, and Yujue Wang. Nebula graph: An open source distributed graph database. *arXiv preprint arXiv:2206.07278*, 2022.

[245] Ying Hu, Seema Sundara, Timothy Chorma, and Jagannathan Srinivasan. Supporting rfid-based item tracking applications in oracle dbms using a bitmap datatype. In *Proceedings of the 31st international conference on Very large data bases*, pages 1140–1151, 2005.

[246] Daniel Ritter, Luigi Dell'Aquila, Andrii Lomakin, and Emanuele Tagliaferri. Orientdb: A nosql, open source mmdms. In *BICOD*, pages 10–19, 2021.

[247] Matteo Lissandrini, Martin Brugnara, and Yannis Velegrakis. Beyond macrobenchmarks: microbenchmark-based graph database evaluation. *Proceedings of the VLDB Endowment*, 12(4):390–403, 2018.

[248] Alin Deutsch, Yu Xu, Mingxi Wu, and Victor Lee. Tigergraph: A native mpp graph database. *arXiv preprint arXiv:1901.08248*, 2019.

[249] Typedb. `https://typedb.com/`. Accessed: 2024-11-14.

[250] Jaime R Carbonell. Ai in cai: An artificial-intelligence approach to computer-assisted instruction. *IEEE transactions on man-machine systems*, 11(4):190–202, 1970.

[251] Richard M Ryan and Edward L Deci. Intrinsic and extrinsic motivation from a self-determination theory perspective: Definitions, theory, practices, and future directions. *Contemporary educational psychology*, 61:101860, 2020.

[252] S Ferilli, Laurie Loop, W Rankin, and P Trafford. Introducing keplair-a platform for independent learners. In *EDULEARN21 Proceedings*, pages 9638–9647. IATED, 2021.

[253] Stefano Ferilli, Berardina De Carolis, and Domenico Redavid. An intelligent agent architecture for smart environments. In *International Symposium on Methodologies for Intelligent Systems*, pages 324–330. Springer, 2015.

[254] Ieee standard for learning object metadata. *IEEE Std 1484.12.1-2020*, pages 1–50, 2020.

[255] Stefano Ferilli and Liza Loop. Toward reasoning-based recommendation of library items - a case study on the e-learning domain. In *Proceedings of the 18th Italian Research Conference on Digital Libraries, IRCDL 2022*, volume 3160, 2022.

[256] Andrew Gardner, Jinko Kanno, Christian A Duncan, and Rastko Selmic. Measuring distance between unordered sets of different sizes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 137–143, 2014.

[257] Kathy J Horadam and Michael A Nyblom. Distances between sets based on set commonality. *Discrete Applied Mathematics*, 167:310–314, 2014.

[258] Houssem Gasmi and Abdelaziz Bouras. Ontology-based education/industry collaboration system. *IEEE Access*, 6:1362–1371, 2017.

[259] Tarcisio Mendes de Farias, Ana Roxin, and Christophe Nicolle. Swrl rule-selection methodology for ontology interoperability. *Data & Knowledge Engineering*, 105:53–72, 2016.

[260] Virginie Fortineau, Thomas Paviot, Ludovic Louis-Sidney, and Samir Lamouri. Swrl as a rule language for ontology-based models in power plant design. In *Product Lifecycle Management. Towards Knowledge-Rich Enterprises: IFIP WG 5.1 International Conference, PLM 2012, Montreal, QC, Canada, July 9-11, 2012, Revised Selected Papers 9*, pages 588–597. Springer, 2012.

[261] William Villegas-Ch and Joselin García-Ortiz. Enhancing learning personalization in educational environments through ontology-based knowledge representation. *Computers*, 12(10):199, 2023.

[262] Stefano Ferilli. Holistic graph-based representation and ai for digital library management. In *International Conference on Theory and Practice of Digital Libraries*, pages 485–489. Springer, 2022.

[263] Stefano Ferilli, Domenico Redavid, and Davide Di Pierro. Holistic graph-based document representation and management for open science. *International Journal on Digital Libraries*, 24(4):205–227, 2023.

[264] Dominik Schlechtweg, Sabine Schulte im Walde, and Stefanie Eckmann. Diachronic usage relatedness (durel): A framework for the annotation of lexical semantic change. *arXiv preprint arXiv:1804.06517*, 2018.

[265] Ethan Allen Andrews and Charlton Thomas Lewis. *A Latin dictionary founded on Andrews' edition of Freund's Latin dictionary*. Clarendon Press, 1896.

[266] Greta Franzini, Andrea Peverelli, Paolo Ruffolo, Marco Passarotti, Helena Sanna, Edoardo Signoroni, Viviana Ventura, and Federica Zampedri. Nunc est aestimandum: Towards an evaluation of the latin wordnet. In *CLiC-it*, 2019.

[267] Christiane Fellbaum. Wordnet: An electronic lexical database. *MIT Press google schola*, 2:678–686, 1998.

[268] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[269] Sascha Schimke, Claus Vielhauer, and Jana Dittmann. Using adapted levenshtein distance for on-line signature authentication. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 2, pages 931–934. IEEE, 2004.

[270] Emanuele Pianta, Luisa Bentivogli, and Christian Girardi. Multiwordnet: developing an aligned multilingual database. In *First international conference on global WordNet*, pages 293–302, 2002.

[271] Elizabeth Closs Traugott. On regularity in semantic change. 1985.

[272] Paul J Hopper et al. On some principles of grammaticization. *Approaches to grammaticalization*, 1:17–35, 1991.

[273] Matthew Horridge and Sean Bechhofer. The owl api: A java api for owl ontologies. *Semantic web*, 2(1):11–21, 2011.

[274] Eva Savina Malinverni, Berardo Naticchia, Jose Luis Lerma Garcia, Alban Gorreja, Joaquin Lopez Uriarte, and Francesco Di Stefano. A semantic graph database for the interoperability of 3d gis data. *Applied Geomatics*, pages 1–14, 2020.

[275] Stephan Mennicke. Modal schema graphs for graph databases. In Alberto H. F. Laender, Barbara Pernici, Ee-Peng Lim, and José Palazzo M. de Oliveira, editors, *Conceptual Modeling*, pages 498–512, Cham, 2019. Springer.

[276] Shuai Ma, Yang Cao, Wenfei Fan, Jinpeng Huai, and Tianyu Wo. Strong simulation: Capturing topology in graph pattern matching. *ACM Trans. Database Syst.*, 39(1), jan 2014.

[277] Stephan Mennicke. *Non-Standard Semantics for Graph Query Languages*. PhD thesis, Braunschweig University of Technology, Germany, 2020.

[278] M. Henzinger, T. Henzinger, and P. Kopke. Computing simulations on finite and infinite graphs. In *Proceedings of the IEEE 36th Annual Symposium on Foundations of Computer Science*, page 453, Los Alamitos, CA, USA, oct 1995. IEEE Computer Society.

[279] Raffaella Gentilini, Carla Piazza, and Alberto Policriti. From bisimulation to simulation: Coarsest partition problems. *Journal of Automated Reasoning*, 31:73–103, 2003.

[280] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[281] Francesco Ranzato and Francesco Tapparo. Generalizing the paige–tarjan algorithm by abstract interpretation. *Information and Computation*, 206(5):620–651, 2008.

[282] Alexander Schätzle, Antony Neu, Georg Lausen, and Martin Przyjaciel-Zablocki. Large-scale bisimulation of rdf graphs. In *Proceedings of the Fifth Workshop on Semantic Web Information Management*, pages 1–8, 2013.

[283] Jonathan Lemaitre and Jean-Luc Hainaut. Transformation-based framework for the evaluation and improvement of database schemas. In *Advanced Information Systems Engineering: 22nd International Conference, CAiSE 2010, Hammamet, Tunisia, June 7-9, 2010. Proceedings 22*, pages 317–331. Springer, 2010.

[284] Dana Angluin, Michael Frazier, and Leonard Pitt. Learning conjunctions of horn clauses. *Machine Learning*, 9:147–164, 1992.

[285] Zhengyu Xu, Sa Dai, and JJ Garcia-Luna-Aceves. A more efficient distance vector routing algorithm. In *MILCOM 97 MILCOM 97 Proceedings*, volume 2, pages 993–997. IEEE, 1997.

[286] Cristhian AD Deagustini, Santiago E Fulladoza Dalibón, Sebastián Gottifredi, Marcelo A Falappa, Carlos I Chesñevar, and Guillermo R Simari. Relational databases as a massive information source for defeasible argumentation. *Knowledge-Based Systems*, 51:93–109, 2013.

[287] Hengfei Li, Nir Oren, and Timothy J Norman. Probabilistic argumentation frameworks. In *International workshop on theorie and applications of formal argumentation*, pages 1–16. Springer, 2011.

[288] Ronald Fagin and Joseph Y Halpern. Uncertainty, belief, and probability 1. *Computational Intelligence*, 7(3):160–173, 1991.

[289] Stefano Bistarelli, Francesco Santini, et al. Weighted argumentation. *FLAP*, 8(6):1589–1622, 2021.

[290] Stefano Ferilli, Fabio Leuzzi, et al. An analysis of the avetrana murder case through abstract argumentation. In *AI³@ AI* IA*, pages 30–44, 2019.

[291] Vladimir A Uspensky. Kolmogorov and mathematical logic. *The Journal of Symbolic Logic*, 57(2):385–412, 1992.

[292] Chunchun Zhao and Sartaj Sahni. String correction using the damerau-levenshtein distance. *BMC bioinformatics*, 20:1–28, 2019.

[293] Ramiro de Vasconcelos dos Santos Júnior, João Vitor Venceslau Coelho, Nelio Alessandro Azevedo Cacho, and Daniel Sabino Amorim de Araújo. A criminal macrocause classification model: An enhancement for violent crime analysis considering an unbalanced dataset. *Expert Systems with Applications*, 238:121702, 2024.

[294] Barry De Ville. Decision trees. *Wiley Interdisciplinary Reviews: Computational Statistics*, 5(6):448–455, 2013.

[295] Hélène Verhaeghe, Quentin Cappart, Gilles Pesant, and Claude-Guy Quimper. Learning precedences for scheduling problems with graph neural networks. In *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

Appendix A

Appendix

A.1 Graph Connectivity

```
measure_execution_time(Query, Time) :-
    get_time(Start),           % Get the current time
       before executing the query
    call(Query),               % Execute the query
    get_time(End),             % Get the current time after
        executing the query
    Time is End - Start.       % Calculate the elapsed time


member(X, [X|_]).
member(X, [_|T]) :- !, member(X, T).

improved_conn(X, X).
improved_conn(X, Y) :-
    !,
    improved_conn(X, Y, []).

improved_conn(X, Y, _) :-
    improved_arcs(X, L),
    member(Y, L).
```

```
19 improved_conn(X, Y, Visited) :-
20     improved_arcs(X, L),
21     member(Z, L),
22     Z \= Y,
23     \+ member(Z, Visited),
24     format('Verifying improved_conn(~w, ~w)~n', [Z, Y]),
25     improved_conn(Z, Y, [Z|Visited]).
26
27 conn(X, X).
28 conn(X, Y) :- !, conn(X, Y, []).
29
30 conn(X, Y, _) :-
31     arcs(X, L),
32     member(Y, L).
33 conn(X, Y, Visited) :-
34     arcs(X, L),
35     member(Z, L),
36     Z \= Y,
37     \+ member(Z, Visited),
38     format('Verifying conn(~w, ~w)~n', [Z, Y]),
39     conn(Z, Y, [Z|Visited]).
40
41 create_pairs(_, [], []).
42 create_pairs(X, [Y|L], [(X, Y)|Pairs]) :-
43     create_pairs(X, L, Pairs).
44
45 execute_all :-
46     findall(Y, (node(Y, _)), UniqueYs),
47     create_pairs(3, UniqueYs, Pairs),
48     execute_pairs(Pairs).
49
50 execute_pairs([]).
```

```
execute_pairs([(X, Y) | Rest]) :-
    once(conn(X, Y)),
    format('Verifying conn(~w, ~w)~n', [X, Y]),
    execute_pairs(Rest).

measure_all_execution_times :-
    findall(Y, (node(Y, _)), UniqueYs),
    create_pairs(3, UniqueYs, Pairs),
    measure_pairs_execution_time(Pairs).

measure_pairs_execution_time([]).
measure_pairs_execution_time([(X, Y) | Rest]) :-
    measure_execution_time(once(conn(X, Y)), Time),
    measure_execution_time(once(improved_conn(X, Y)),
       ImprovedTime),
    format('~w, ~w~n', [Time, ImprovedTime]),
    measure_pairs_execution_time(Rest).
```

Listing A.1: ASP Graph Connnectivity

A.2 Node Similarity

```
damerau_levenshtein([], [], 0) :- !.
damerau_levenshtein(X, [], Cost) :- length(X, Cost), !.
damerau_levenshtein([], Y, Cost) :- length(Y, Cost), !.
damerau_levenshtein([X|XS], [X|YS], Cost) :-
    damerau_levenshtein(XS, YS, Cost), !.
damerau_levenshtein([X|XS], [Y|YS], Cost) :-
    X \= Y,
    damerau_levenshtein(XS, YS, SubstitutionCost),
    damerau_levenshtein(XS, [Y|YS], DeletionCost),
    damerau_levenshtein([X|XS], YS, InsertionCost),
```

```
    Cost is min(SubstitutionCost + 1, min(DeletionCost + 1,
        InsertionCost + 1)), !.
damerau_levenshtein([X1, X2|XS], [Y1, Y2|YS], Cost) :-
    X1 = Y2, X2 = Y1,
    damerau_levenshtein(XS, YS, TranspositionCost),
    Cost is TranspositionCost + 1, !.

int_distance(V1, V2, D) :-
        D is abs(V1 - V2).

split_date(DateString, Year, Month, Day) :-
    split_string(DateString, "/", "", [YearString,
       MonthString, DayString]),
    number_string(Year, YearString),
    number_string(Month, MonthString),
    number_string(Day, DayString).

month_length(1,31).
month_length(2,28).
month_length(3,31).
month_length(4,30).
month_length(5,31).
month_length(6,30).
month_length(7,31).
month_length(8,31).
month_length(9,30).
month_length(10,31).
month_length(11,30).
month_length(12,31).

days_month(0, Days, Days) :-
        !.
```

```
41 days_month(Month, DaysAcc, Days) :-
42         month_length(Month, L),
43         UpdatedD is DaysAcc + L,
44         PreviousMonth is Month - 1,
45         days_month(PreviousMonth, UpdatedD, Days).
46
47 this_leap(Leaps, UpdatedLeaps, Year, Month, _) :-
48         Mod is Year mod 4,
49         Mod == 0,
50         Month > 2,
51         UpdatedLeaps is Leaps + 1.
52
53 this_leap(Leaps, UpdatedLeaps, Year, Month, Day) :-
54         Year mod 4 == 0,
55         Month == 2,
56         Day == 29,
57         UpdatedLeaps is Leaps + 1.
58
59 this_leap(Leaps, Leaps, Year, Month, Day) :-
60         UpdatedLeaps is Leaps + 1,
61         \+ this_leap(Leaps, UpdatedLeaps, Year, Month, Day)
              .
62
63 date_distance(D1, D2, D) :-
64         split_date(D1, Year1, Month1, Day1),
65         split_date(D2, Year2, Month2, Day2),
66         PreviousMonth1 is Month1 - 1,
67         PreviousMonth2 is Month2 - 1,
68         days_month(PreviousMonth1, 0, DaysMonth1),
69         days_month(PreviousMonth2, 0, DaysMonth2),
70         Leaps1 is (Year1 - 1901) / 4,
71         Leaps2 is (Year2 - 1901) / 4,
```

```
72          this_leap(Leaps1, UpdatedLeaps1, Year1, Month1,
               Days1),
73          this_leap(Leaps2, UpdatedLeaps2, Year2, Month2,
               Days2),
74          Days1 is Year1 * 365 + UpdatedLeaps1 + DaysMonth1 +
                Day1,
75          Days2 is Year2 * 365 + UpdatedLeaps2 + DaysMonth2 +
                Day2,
76          D is abs(Days1 - Days2).
77
78  categorical_distance(C1, C2, D) :-
79          C1 == C2,
80          D is 0.
81
82  categorical_distance(C1, C2, D) :-
83          C1 \== C2,
84          D is 1.
85
86  ordered_distance(O, L1, L2, D) :-
87          order(O, L1, V1),
88          order(O, L2, V2),
89          D is abs(V1 - V2).
90
91  string_distance(S1, S2, D) :-
92          atom_chars(S1, L1),
93          atom_chars(S2, L2),
94          damerau_levenshtein(L1, L2, D), !.
95
96  node_int_distance(_, _, [], DInt, DInt) :-
97      !.
98  node_int_distance(N1, N2, [P | IntProperties], DAcc, DInt)
       :-
```

```
99      property(N1, P, V1),
100     property(N2, P, V2),
101     int_distance(V1, V2, D),
102     UpdatedD is DAcc + D,
103     node_int_distance(N1, N2, IntProperties, UpdatedD, DInt
          ).
104
105 node_date_distance(_, _, [], DDate, DDate) :-
106     !.
107 node_date_distance(N1, N2, [P | DateProperties], DAcc,
      DDate) :-
108     property(N1, P, D1),
109     property(N2, P, D2),
110     date_distance(D1, D2, D),
111     UpdatedD is DAcc + D,
112     node_int_distance(N1, N2, DateProperties, UpdatedD,
          DDate).
113
114 node_categorical_distance(_, _, [], DCategorical,
      DCategorical) :-
115         !.
116 node_categorical_distance(N1, N2, [P |
      CategoricalProperties], DAcc, DCategorical) :-
117         property(N1, P, C1),
118         property(N2, P, C2),
119         categorical_distance(C1, C2, D),
120         UpdatedD is DAcc + D,
121         node_categorical_distance(N1, N2,
              CategoricalProperties, UpdatedD, DCategorical).
122
123 node_ordered_distance(_, _, [], DOrdered, DOrdered) :-
124         !.
```

```
125 node_ordered_distance(N1, N2, [P | OrderedProperties], DAcc
    , DOrdered) :-
126         property(N1, P, L1),
127         property(N2, P, L2),
128         L1 \= L2,
129         ordered_distance(P, L1, L2, D),
130         UpdatedD is DAcc + D,
131         node_ordered_distance(N1, N2, OrderedProperties,
              UpdatedD, DOrdered).
132
133 node_string_distance(_, _, [], DString, DString) :-
134     !.
135 node_string_distance(N1, N2, [P | StringProperties], DAcc,
    DString) :-
136     property(N1, P, V1),
137     property(N2, P, V2),
138     string_distance(V1, V2, D),
139     UpdatedD is DAcc + D,
140     node_string_distance(N1, N2, StringProperties, UpdatedD
          , DString),
141     !.
142
143 node_distance(N1, N2, AlfaInt, AlfaDate, AlfaCategorical,
    AlfaOrdered, AlfaString, Properties, Distance) :-
144         findall(P1, property(N1, P1, _), Properties1),
145         findall(P2, property(N2, P2, _), Properties2),
146         findall(P, (member(P, Properties1), member(P,
              Properties2)), Properties),
147         findall(P, (member(P, Properties), type(P, int)),
              IntProperties),
148         findall(P, (member(P, Properties), type(P, date)),
              DateProperties),
```

```
149         findall(P, (member(P, Properties), type(P,
                categorical)), CategoricalProperties),
150         findall(P, (member(P, Properties), type(P, ordered)
                ), OrderedProperties),
151         findall(P, (member(P, Properties), type(P, string))
                , StringProperties),
152         node_int_distance(N1, N2, IntProperties, 0, DInt),
153         node_date_distance(N1, N2, DateProperties, 0, DDate
                ),
154         node_categorical_distance(N1, N2,
                CategoricalProperties, 0, DCategorical),
155         node_ordered_distance(N1, N2, OrderedProperties, 0,
                 DOrdered),
156         node_string_distance(N1, N2, StringProperties, 0,
                DString),
157         Numerator is AlfaInt * DInt + AlfaDate * DDate +
                AlfaCategorical * DCategorical + AlfaOrdered *
                DOrdered + AlfaString * DString,
158         length(IntProperties, LInt),
159         length(DateProperties, LDate),
160         length(CategoricalProperties, LCategorical),
161         length(OrderedProperties, LOrdered),
162         length(StringProperties, LString),
163         Divisor is LInt * AlfaInt + LDate * AlfaDate +
                LCategorical * AlfaCategorical + LOrdered *
                AlfaOrdered + LString * AlfaString,
164         Distance is Numerator / Divisor.
165
166 child(X, X).
167 child(X, Y) :-
168         X \== Y,
169         isA(X, Z),
```

```
170         child(Z, Y).
171
172 distance(N1, N2, Distance) :-
173         AlfaInt is 1,
174         AlfaDate is 0.1,
175         AlfaCategorical is 1,
176         AlfaOrdered is 1,
177         AlfaString is 1,
178         node(N1, Entity1),
179         node(N2, Entity2),
180         child(Entity1, Entity2),
181         !,
182         findall(X, attribute(X, Entity2), Properties),
183         node_distance(N1, N2, AlfaInt, AlfaDate,
               AlfaCategorical, AlfaOrdered, AlfaString,
               Properties, Distance).
184
185 distance(N1, N2, Distance) :-
186         AlfaInt is 1,
187         AlfaDate is 0.1,
188         AlfaCategorical is 1,
189         AlfaOrdered is 1,
190         AlfaString is 1,
191         node(N1, Entity1),
192         node(N2, Entity2),
193         child(Entity2, Entity1),
194         !,
195         findall(X, attribute(X, Entity1), Properties),
196         node_distance(N1, N2, AlfaInt, AlfaDate,
               AlfaCategorical, AlfaOrdered, AlfaString,
               Properties, Distance).
```

Listing A.2: ASP Node Similarity

Definitions

Ringraziamenti