

# Metadata of the Book that will be visualized online

Book Title	Semantic Search over the Web	
Book SubTitle		
Copyright Year	2012	
Copyright Holder	Springer-Verlag Berlin Heidelberg	
Editor	Family Name	De Virgilio
	Particle	
	Given Name	<b>Roberto</b>
	Suffix	
	Division	, Dipartimento di Informatica
	Organization	Università degli Studi Roma Tre
	Address	Via della Vasca Navale 79, 00146, Roma, Italy
	Email	rde79@yahoo.com
Editor	Family Name	Guerra
	Particle	
	Given Name	<b>Francesco</b>
	Suffix	
	Division	e Reggio Emilia, Dipartimento di Economia Aziendale
	Organization	Università degli Studi di Modena
	Address	Via le Berengario, 51, 41100, Modena, Italy
	Email	francesco.guerra@unimore.it
Editor	Family Name	Velegakis
	Particle	
	Given Name	<b>Yannis</b>
	Suffix	
	Division	
	Organization	Università degli Studi di Trento
	Address	Via Sommarive 14, 38123, Trento, Italy
	Email	velgias@disi.unitn.eu

## Data-Centric Systems and Applications

---

*Series Editors*

M.J. Carey  
S. Ceri

*Editorial Board*

P. Bernstein  
U. Dayal  
C. Faloutsos  
J.C. Freytag  
G. Gardarin  
W. Jonker  
V. Krishnamurthy  
M.-A. Neimat  
P. Valduriez  
G. Weikum  
K.-Y. Whang  
J. Widom

UNCORRECTED PROOF

Roberto De Virgilio • Francesco Guerra  
Yannis Velegrakis

Editors

# Semantic Search over the Web

UNCORRECTED PROOF



Springer

*Editors*

Roberto De Virgilio  
Dipartimento di Informatica  
e Automazione  
Università degli Studi Roma Tre  
Roma  
Italy

Francesco Guerra  
Dipartimento di Economia Aziendale  
Università degli Studi di Modena  
e Reggio Emilia  
Modena  
Italy

Yannis Velegrakis  
Università degli Studi di Trento  
Trento  
Italy

ISBN 978-3-642-25007-1      e-ISBN 978-3-642-25008-8  
DOI 10.1007/978-3-642-25008-8  
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: "PCN Applied for"

ACM Computing Classification: H.3, I.2

© Springer-Verlag Berlin Heidelberg 2012

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Printed on acid-free paper

Springer is part of Springer Science+Business Media ([www.springer.com](http://www.springer.com))

# Introduction

1

The Web has become the world's largest database with search being the main 2 tool that enables organization and individuals to exploit its huge amounts of 3 information that is freely offering. Thus, having a successful mechanism for 4 finding and retrieving the most relevant information to a task at hand is of major 5 importance. Traditionally, Web search has been based on textual and structural 6 similarity. Given the set of keywords that comprise a query, the goal is to identify 7 the documents containing all these keywords (or as many as possible). Additional 8 information such as information from logs, references from authorities, popularity, 9 and personalization has been extensively used to further improve the accuracy. 10 However, one of the dimensions that has not been captured to its full extent is that 11 of *semantics*, that is, fully understanding the meaning of the words in a query and in 12 a document. Combining search and semantics gives birth to the idea of the *semantic* 13 *search*. Semantic search can be described in a sentence as the effort of improving 14 the accuracy of the search process by understanding the context and limiting the 15 ambiguity. 16

AQ1      The idea of the semantic Web is based on this goal and aims at making the 17 semantics of the Web content machine understandable. To do so, a number of 18 different technologies that allowed for richer modeling of the Web resources, along- 19 side annotations describing their semantics, have been introduced. Furthermore, the 20 semantic Web went on to create associations between different representations of the 21 same real-world entity. These associations are either explicitly specified or derived 22 off-line and then remain static. They allow data from many different sources to 23 be interlinked, giving birth to the so-called linked open data cloud. Nevertheless, 24 semantics have yet to fully penetrate existing data management solutions and 25 become an integral part in information retrieval, analysis, integration, and data 26 exchange techniques. 27

Unfortunately, the generic idea of semantic search has remained in its infancy. 28 Existing solutions are either search engines that simply index the semantic Web data, 29 like Sindice, or the traditional search engines enhanced with some basic form of 30 synonym exploitation, as supported by Google and Bing. Semantic search is about 31 using the semantics of the query terms instead of the terms themselves. This means 32

using synonyms and related terms, providing additional materials in the answer that 33  
may be related to elements already in the result, searching not only in the content 34  
but also in the semantic annotations of the data, exploiting ontological knowledge 35  
through advanced reasoning techniques, treating the query as a natural language 36  
expression, clustering the results, offering faced browsing, etc. 37

All the above mean that there are currently numerous opportunities to exploit 38  
in the area of semantic search on the Web. In this work, we try to give a generic 39  
overview of the works that have been done in the field and in other related areas. 40  
However, the work should definitely not be considered as a survey. It is simply 41  
intended to provide the reader with a taste of the many different aspects of the 42  
problem and go deep in some specific technologies and solutions. 43

The book is divided into three parts. The first part introduces the notion of the 44  
Web of Data. It describes the different types of data that exist, their topology, and 45  
their storing and indexing techniques. It also shows how semantic links between the 46  
data can be automatically derived. 47

The second part is dedicated specifically to Web search. It presents different 48  
kinds of search, such as the exploratory or the path-oriented, alongside methods 49  
for efficiently implementing them. It talks about the problem of interactive query 50  
construction and also about the understanding of the keyword query semantics. 51  
Other topics include the use of uncertainty in query answering or the exploitation of 52  
ontologies. The second part concludes with some reference to Mashup technologies 53  
and the way they are affected by the semantics. 54

The theme of the third part of the book is Linked Data and, more specifically, 55  
how recommender system ideas can be used in the case of linked data management 56  
alongside techniques for efficient query answering. 57

Rome, Italy  
Modena, Italy  
Trento, Italy

*Roberto De Virgilio* 58  
*Francesco Guerra* 59  
*Yannis Velegrakis* 60

# Contents

1

## Part I Introduction to Web of Data

<b>1</b>	<b>Topology of the Web of Data .....</b>	<b>3</b>	<b>2</b>
	Christian Bizer, Pablo N. Mendes, and Anja Jentzsch		3
<b>2</b>	<b>Storing and Indexing Massive RDF Datasets .....</b>	<b>29</b>	<b>4</b>
	Yongming Luo, François Picalausa, George H.L. Fletcher, Jan Hidders, and Stijn Vansumeren		5
			6
<b>3</b>	<b>Designing Exploratory Search Applications upon Web Data Sources .....</b>	<b>59</b>	<b>7</b>
	Marco Brambilla and Stefano Ceri		8
			9

## Part II Search over the Web

<b>4</b>	<b>Path-Oriented Keyword Search Query over RDF .....</b>	<b>79</b>	<b>10</b>
	Roberto De Virgilio, Paolo Cappellari, Antonio Maccioni, and Riccardo Torlone		11
			12
<b>5</b>	<b>Interactive Query Construction for Keyword Search on the Semantic Web .....</b>	<b>107</b>	<b>13</b>
	Gideon Zenz, Xuan Zhou, Enrico Minack, Wolf Siberski, and Wolfgang Nejdl		14
			15
			16
<b>6</b>	<b>Understanding the Semantics of Keyword Queries on Relational Data Without Accessing the Instance .....</b>	<b>129</b>	<b>17</b>
	Sonia Bergamaschi, Elton Domnori, Francesco Guerra, Silvia Rota, Raquel Trillo Lado, and Yannis Velegrakis		18
			19
			20
<b>7</b>	<b>Keyword-Based Search over Semantic Data .....</b>	<b>157</b>	<b>21</b>
	Klara Weiand, Andreas Hartl, Steffen Hausmann, Tim Furche, and François Bry		22
			23

<b>8</b>	<b>Semantic Link Discovery over Relational Data</b>	191	24
	Oktie Hassanzadeh, Anastasios Kementsietsidis, Lipyeow Lim, Renée J. Miller, and Min Wang	25	26
<b>9</b>	<b>Embracing Uncertainty in Entity Linking</b>	223	27
	Ekaterini Ioannou, Wolfgang Nejdl, Claudia Niederée, and Yannis Velegrakis	28	29
<b>10</b>	<b>The Return of the Entity-Relationship Model: Ontological Query Answering</b>	30	31
	Andrea Calì, Georg Gottlob, and Andreas Pieris	32	33
<b>11</b>	<b>Linked Data Services and Semantics-Enabled Mashup</b>	281	33
	Devis Bianchini and Valeria De Antonellis	34	
<b>Part III Linked Data Search Engines</b>			
<b>12</b>	<b>A Recommender System for Linked Data</b>	309	35
	Roberto Mirizzi, Azzurra Ragone, Tommaso Di Noia, and Eugenio Di Sciascio	36	37
<b>13</b>	<b>Flint: From Web Pages to Probabilistic Semantic Data</b>	331	38
	Lorenzo Blanco, Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti	39	40
<b>14</b>	<b>Searching and Browsing Linked Data with SWSE*</b>	361	41
	Andreas Harth, Aidan Hogan, Jürgen Umbrich, Sheila Kinsella, Axel Polleres, and Stefan Decker	42	43
<b>Index</b>		415	44

## Contributors

1

<b>Sonia Bergamaschi</b> Dipartimento di Ingegneria dell'Informazione, Università di Modena e Reggio Emilia, Modena, Italy	2 3
<b>Devis Bianchini</b> Department of Electronics for Automation, University of Brescia, Brescia, Italy	4 5
<b>Christian Bizer</b> Web-based Systems Group, Freie Universität Berlin, Berlin, Germany	6 7
<b>Lorenzo Blanco</b> Dipartimento di Informatica e Automazione, Università degli Studi Roma Tre, Rome, Italy	8 9
<b>Marco Brambilla</b> Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy	10 11
<b>Mirko Bronzi</b> Dipartimento di Informatica e Automazione, Università degli Studi Roma Tre, Rome, Italy	12 13
<b>François Bry</b> Institute for Informatics, University of Munich, München, Germany	14
<b>Andrea Calì</b> Department of Computer Science and Information Systems, Birkbeck University of London, London, UK	15 16
<b>Paolo Cappellari</b> Interoperable System Group, Dublin City University, Dublin, Ireland	17 18
<b>Stefano Ceri</b> Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy	19 20
<b>Valter Crescenzi</b> Dipartimento di Informatica e Automazione, Università degli Studi Roma Tre, Rome, Italy	21 22
<b>Valeria De Antonellis</b> Department of Electronics for Automation, University of Brescia, Brescia, Italy	23 24
<b>Roberto De Virgilio</b> University Roma Tre, Rome, Italy	25

<b>Tommaso Di Noia</b>	Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Bari, Italy	26 27
<b>Eugenio Di Sciascio</b>	Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Bari, Italy	28 29
<b>Stefan Decker</b>	Digital Enterprise Research Institute, National University of Ireland, Galway, Ireland	30 31
<b>Elton Domnori</b>	Dipartimento di Ingegneria dell'Informazione, Università di Modena e Reggio Emilia, Modena, Italy	32 33
<b>George H.L. Fletcher</b>	Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands	34 35
<b>Tim Furche</b>	Department of Computer Science and Institute for the Future of Computing, Oxford University, Oxford, UK	36 37
<b>Georg Gottlob</b>	Computing Laboratory, University of Oxford, Oxford, UK	38
	Oxford-Man Institute of Quantitative Finance, University of Oxford, Oxford, UK	39
<b>Francesco Guerra</b>	Dipartimento di Economia Aziendale, Università di Modena e Reggio Emilia, Modena, Italy	40 41
<b>Andreas Harth</b>	Karlsruhe Institute of Technology, Institute AIFB, Karlsruhe, Germany	42 43
<b>Andreas Hartl</b>	Institute for Informatics, University of Munich, München, Germany	44 45
<b>Oktie Hassanzadeh</b>	University of Toronto, Toronto, Ontario, Canada	46
<b>Steffen Hausmann</b>	Institute for Informatics, University of Munich, München, Germany	47 48
<b>Jan Hidders</b>	Faculty of Electrical Engineering Mathematics and Computer Science, Delft University of Technology, Delft, The Netherlands	49 50
<b>Aidan Hogan</b>	Digital Enterprise Research Institute, National University of Ireland, Galway, Ireland	51 52
<b>Ekaterini Ioannou</b>	University Campus – Kounoupidiana, Technical University of Crete, Chania, Greece	53 54
<b>Anja Jentzsch</b>	Web-based Systems Group, Freie Universität Berlin, Berlin, Germany	55 56
<b>Anastasios Kementsietsidis</b>	IBM T.J. Watson Research Center, Hawthorne, NY, USA	57 58
<b>Sheila Kinsella</b>	Digital Enterprise Research Institute, National University of Ireland, Galway, Ireland	59 60

<b>Lipyeow Lim</b> University of Hawaii at Manoa, Honolulu, HI, USA	61
<b>Yongming Luo</b> Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands	62 63
<b>Antonio Maccioni</b> University Roma Tre, Rome, Italy	64
<b>Pablo N. Mendes</b> Web-based Systems Group, Freie Universität Berlin, Berlin, Germany	65 66
<b>Paolo Merialdo</b> Dipartimento di Informatica e Automazione, Università degli Studi Roma Tre, Rome, Italy	67 68
<b>Renée J. Miller</b> University of Toronto, Toronto, Ontario, Canada	69
<b>Enrico Minack</b> L3S Research Center, Hannover, Germany	70
<b>Roberto Mirizzi</b> Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Bari, Italy	71 72
<b>Wolfgang Nejdl</b> L3S Research Center, Hannover, Germany	73
<b>Claudia Niederée</b> L3S Research Center, Hannover, Germany	74
<b>Paolo Papotti</b> Dipartimento di Informatica e Automazione, Università degli Studi Roma Tre, Rome, Italy	75 76
<b>François Picalausa</b> Université Libre de Bruxelles, Brussels, Belgium	77
<b>Andreas Pieris</b> Department of Computer Science, University of Oxford, Oxford, UK	78 79
<b>Axel Polleres</b> Siemens AG Österreich, Vienna, Austria	80
Digital Enterprise Research Institute, National University of Ireland, Galway, Ireland	81 82
<b>Azzurra Ragone</b> Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Bari, Italy	83 84
Exprivia S.p.A., Molfetta, BA, Italy	85
<b>Silvia Rota</b> Dipartimento di Ingegneria dell'Informazione, Università di Modena e Reggio Emilia, Modena, Italy	86 87
<b>Wolf Siberski</b> L3S Research Center, Hannover, Germany	88
<b>Riccardo Torlone</b> University Roma Tre, Rome, Italy	89
<b>Raquel Trillo</b> Informatica e Ing. Sistemas, Zaragoza, Spain	90
<b>Jürgen Umbrich</b> Digital Enterprise Research Institute, National University of Ireland, Galway, Ireland	91 92
<b>Stijn Vansumeren</b> Université Libre de Bruxelles, Brussels, Belgium	93

<b>Yannis Velegrakis</b>	University of Trento, Trento, Italy	94
<b>Min Wang</b>	HP Labs China, Beijing, China	95
<b>Klara Weiand</b>	Institute for Informatics, University of Munich, München, Germany	96
<b>Gideon Zenz</b>	L3S Research Center, Hannover, Germany	98
<b>Xuan Zhou</b>	Renmin University of China, Beijing, China	99

UNCORRECTED PROOF

AUTHOR QUERY

AQ1. Please confirm the changes in the sentence “To do so, a number...”

UNCORRECTED PROOF

**Part I<sub>1</sub>**  
**Introduction to Web of Data<sub>2</sub>**

UNCORRECTED PROOF

# Chapter 1

## Topology of the Web of Data

Christian Bizer, Pablo N. Mendes, and Anja Jentzsch

### 1.1 Introduction

The degree of structure of Web content is a determining factor for the types of functionality that search engines can provide. The more well structured the Web content is, the easier it is for search engines to understand Web content and provide advanced functionality, such as faceted filtering or the aggregation of content from multiple Web sites based on this understanding.

Today, most Web sites are published as unstructured or semistructured data that are generated from structured data that are stored in relational databases. Thus, it does not require too much extra effort for Web sites to publish these structured data directly on the Web and thus help search engines to understand Web content and provide improved functionality.

An early approach to realize this idea and help search engines to understand Web content is Microformats,<sup>1</sup> a technique for marking up structured data about specific types on entities—such as tags, blog posts, people, or reviews—within HTML pages. As Microformats are focused on a few entity types, the World Wide Web Consortium (W3C) started in 2004 to standardize RDFa [1] as an alternative, more generic language for embedding any type of data into HTML pages.

Today, major search engines such as Google, Yahoo, and Bing extract Microformat and RDFa data describing products, reviews, persons, events, and recipes from Web pages and use the extracted data to improve the user's search experience. The search engines have started to aggregate structured data from different Web sites and augment their search results with these aggregated

---

<sup>1</sup><http://microformats.org/>

AQ1 C. Bizer (✉) · P.N. Mendes · A. Jentzsch  
Web-based Systems Group, Freie Universität Berlin, Garystr. 21-14195 Berlin, Germany  
e-mail: [christian.bizer@fu-berlin.de](mailto:christian.bizer@fu-berlin.de); [pablo.mendes@fu-berlin.de](mailto:pablo.mendes@fu-berlin.de); [anja.jentzsch@fu-berlin.de](mailto:anja.jentzsch@fu-berlin.de)

information units in the form of rich snippets which combine, for instance, data 26  
describing a product with reviews of the product from different sites.<sup>2</sup> Another 27  
major consumer of RDFa data is Facebook which imports structured data from 28  
external Web sites once a user clicks a Facebook *Like*-button on one of these sites.<sup>3</sup> 29

The support of Microformats and RDFa by major data consumers, such as 30  
Google, Yahoo!, Microsoft, and Facebook, has led to a sharp increase in the number 31  
of Web sites that embed structured data into HTML pages. According to statistics 32  
presented by Yahoo!, the number of Web pages containing RDFa data has increased 33  
by 510% between 2009 and 2010. As of October 2010, 430 million Web pages 34  
contained RDFa markup, while over 300 million of pages contained microformat 35  
data [34]. In a recent move, Google, Yahoo!, and Microsoft have jointly agreed 36  
on a set of vocabularies for describing over 200 different types of entities. Entity 37  
descriptions that are expressed using these vocabularies and are embedded into 38  
HTML pages using the microdata syntax<sup>4</sup> will be equally understood by all three 39  
search engines. This move toward standardization is likely to further increase the 40  
amount of structured data being published on the Web. 41

AQ2

Parallel to the different techniques to embed structured data into HTML pages, 42  
a set of best practices for publishing structured data directly on the Web has 43  
got considerable traction: Linked Data [4, 10]. The Linked Data best practices 44  
are implemented today by hundreds of data providers including the UK and US 45  
governments, media companies like the BBC and the New York Times, various 46  
national libraries, and a worldwide community of research institutions. While 47  
microformats, RDFa, and microdata focus on publishing descriptions of single, 48  
isolated entities on the Web, Linked Data focuses on making the relationships 49  
between entities explicit. Data describing relationships are published in the form 50  
of RDF Links. RDF Links can span different Web servers and connect all data that 51  
are published as Linked Data into a single global data graph [5]. As of July 2011, 52  
this data graph consists of over 31 billion edges (RDF triples) [12]. 53

AQ3

This chapter gives an overview of the topology of the Web of Data that has been 54  
created by publishing data on the Web using the Microformats, RDFa, Microdata, 55  
and Linked Data publishing techniques. Section 1.2 discusses Microformats and 56  
provides statistics about the Microformats deployment on the Web. Sections 1.3 57  
and 1.4 cover RDFa and Microdata. Section 1.5 discusses Linked Data and gives 58  
an overview of the Linked Data deployment on the Web. For each of the four 59  
techniques, we: 60

AQ4

1. Summarize the main features and give an overview of the history of the technique 61
2. Provide a syntax example which shows how data describing a person are 62  
published on the Web using the technique 63
3. Present deployment statistics showing the amounts and types of data currently 64  
published using the specific technique 65

<sup>2</sup><http://www.google.com/support/webmasters/bin/answer.py?answer=1095551>

<sup>3</sup><http://developers.facebook.com/docs/opengraph/>

<sup>4</sup><http://www.whatwg.org/specs/web-apps/current-work/multipage/microdata.html>

The syntax examples highlight how the different techniques handle (1) the identification of entities; (2) the representation of the type information—e.g., that the entity is a person; (3) the representation of literal property values, such as the name of the person; and (4) the representation of relationships between entities, such as that Peter Smith knows Paula Jones.

In order to provide an entry point for experimentation as well as for the evaluation of search engines that facilitate Web data, Sect. 1.6 gives an overview of large-scale datasets that have been crawled from the Web of Data and are publicly available for download.

## 1.2 Microformats

75

AQ5 Microformats<sup>5</sup> (also referred to as  $\mu F$ ) are community-driven vocabulary agreements for providing semantic markup on Web pages. The motto of the Microformats community is “designed for humans first, machines second.” Each Microformat defines a vocabulary and a syntax for applying the vocabulary to describe the content on Web pages. A microformat’s syntax commonly specifies which properties are required or optional and which classes should be nested under one another.

Microformats emerged as a community effort, in contrast to other semantic mark-up technologies which sought the route of a standardization body. Early contributors to Microformats include Kevin Marks, Tantek Çelik, and Mark Pilgrim, among others. The first implementations of Microformats date back to 2003,<sup>6</sup> when they were mostly found in the blogosphere. One early microformat, the XHTML Friends Network (XFN),<sup>7</sup> allowed one to represent human relationships between blog owners through the use of marked-up hyperlinks between blogs. In 2004 followed XOXO for representing outlines in XHTML; VoteLinks for specifying endorsement and disagreement links; *rel-license* for embedding links to licenses; *hCard* for representing people, companies, and organizations; and *hCalendar* for representing calendars and events. The year of 2005 saw the introduction of *hReview*, for embedding reviews (e.g., of products or services) in Web documents. Many other microformats have been introduced since then, amounting to more than 25 among

AQ7 drafts and specifications available from <http://microformats.org>.

It is argued that the simplistic approach offered by Microformats eases the learning curve and therefore lowers the entry barrier for newcomers. On the other hand, due to the lack of unified syntax for all microformats, consuming structured data from microformats requires the development of specialized parsers for each format. This is a reflection of the Microformats approach to address specific use

<sup>5</sup><http://microformats.org/>

<sup>6</sup><http://microformats.org/wiki/history-of-microformats>

<sup>7</sup><http://gmpg.org/xfn/>

```
<head profile="http://gmpg.org/xfn/11">
<head profile="http://microformats.org/profile/hcard">
...
<div class="vcard">
  <a class="url fn" href="http://example.com/Peter.html">
    Peter Smith </a>
  <a rel="met acquaintance"
    href="http://example.com/Paula.html">Paula Jones </a>.
</div>
```

**Fig. 1.1** Microformats representation of the example entity description

cases, in contrast to RDFa and Microdata (presented in the following sections) 101  
which support the representation of any kind of data.<sup>8</sup> 102

AQ8

### 1.2.1 *Microformats Syntax*

103

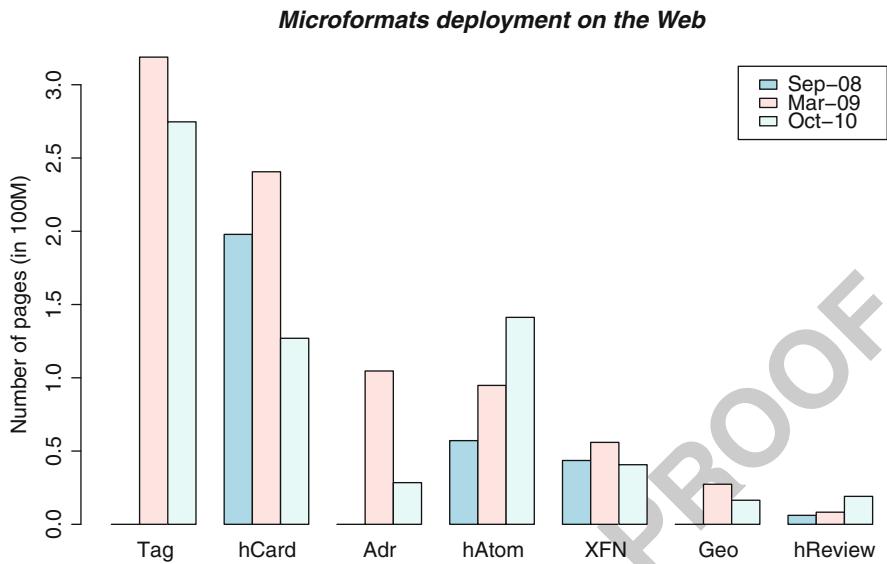
Microformats consist of a definition of a vocabulary (names for classes and 104 properties), as well as a set of rules (e.g., required properties, correct nesting 105 of elements). These rules largely rely on existing HTML/XHTML attributes for 106 inserting markup. One example is the HTML attribute `class`, commonly used as a 107 style sheet selector, which is reused in microformats for describing properties and 108 types of entities. 109

Figure 1.1 shows an HTML snippet containing the Microformat representation 110 of our example data describing Peter Smith. The `vcard` added in line 4 is a root class 111 name indicating the presence of an hCard. Since hCards are used for representing 112 contact information about people, organizations, etc., one could assume that an 113 entity of one of those types is present in that page. The properties of an hCard are 114 represented by class names added to HTML elements inside the hCard. The example 115 in Fig. 1.1 shows the properties `url` (Peter's home page) and `fn` (Peter's full name). 116 The markup also states that Peter knows Paula through the use of the properties 117 `met` and `acquaintance` defined in the XFN microformat. An hCard parser should 118 be aware that the `url` property refers to the value of the `href` attribute, while `fn` 119 refers to the value of the child text of the HTML element. Such parsing instructions 120 are described within each microformat specification. 121

The microformats community encourages mixing microformats and reusing 122 existing formats when creating new ones. Naming conflicts between formats are not 123 handled through the microformats technology, but rather through a social process<sup>9</sup> 124 of creating microformat specifications. 125

<sup>8</sup><http://microformats.org/2011/02/11/whats-next-for-microformats>

<sup>9</sup><http://microformats.org/wiki/process>



**Fig. 1.2** Microformats deployment on the Web, according to [34]

### 1.2.2 Microformats Deployment on the Web

126

In June 2006, Yahoo! Local announced the deployment of hCalendar, hCard, and hReview microformats on almost all business listings, search results, events, and reviews [16]. Yahoo! Europe followed in 2008 with a deployment of ca. 26 million more annotated pages describing merchants in classified listings, resulting on the biggest deployment of the hListing format at that time.

These developments were followed with adoption from the data consuming side. In March 2008, Yahoo! Search announced that it was indexing semantic markup including hCard, hCalendar, hReview, hAtom, and XFN [17]. In May 2009, Google also announced that they would be parsing the hCard, hReview, and hProduct microformats and using them to populate search result pages [20].

Since then, a number of major Web sites have started to support microformats. For instance, Facebook publishes approximately 39 million event pages annotated with hCalendar, Yelp.com adds hReview and hCard to all of their listings (about 56 million businesses), and Wikipedia templates are able to automatically generate microformats such as geo, hCard, and hCalendar markup.<sup>10</sup>

The most recent publicly available statistics about the deployment of Microformats on the Web were produced by Yahoo! Research [34] and presented in January 2011. Figure 1.2 shows the deployment of some of the most common microformats

<sup>10</sup>[http://en.wikipedia.org/wiki/Category:Templates\\_generating\\_microformats](http://en.wikipedia.org/wiki/Category:Templates_generating_microformats)

on the Web. The results are based on an analysis of a sample of 12 billion Web 145  
pages indexed by Yahoo! Search. Results are displayed for samples obtained in 146  
2008, 2009, and 2010. 147

Microformats commonly associated with blogs are prevalent in this sample 148  
(hAtom, tag, hCard). Besides hAtom and hReview, the numbers do not show a trend 149  
of stable increase and, in some cases (e.g., hCard, adr, xfn, geo), show a reduction 150  
on the number of deployments since 2008. The microformats tag, adr, and geo were 151  
not counted in 2008 and therefore are missing. 152

## 1.3 RDFa

153

The resource description framework (RDF) is a simple graph-based data model 154  
which is recommended by the W3C for exchanging structured data on the Web [31]. 155  
RDF represents data in the form of triples, where each triple is composed by an 156  
entity identifier, a property, and a value for this property. 157

The RDFa [1] serialization format for embedding RDF data into HTML pages 158  
was proposed in 2004 by Mark Birbeck as a way to “make more of the information 159  
that is contained within HTML-family documents available to RDF tools.” The 160  
initial note was presented to the Semantic Web Interest Group in 2004 and developed 161  
jointly with the XHTML 2 Working Group into a W3C Recommendation published 162  
in October 2008. 163

RDFa allows one to embed RDF triples within the HTML *document object model* 164  
(DOM). A detailed introduction to RDFa is given in the W3C RDFa Primer [1]. In 165  
contrast to the microformats approach, where each microformat defines its syntax 166  
and vocabulary, RDFa provides single syntax which can be used together with any 167  
vocabulary or mixture of terms from multiple vocabularies. 168

AQ9

### 1.3.1 RDFa Syntax

169

The RDFa syntax specifies how HTML elements may be annotated with entity 170  
identifiers, entity types, string properties, and relationship properties. Figure 1.3 171  
shows the RDFa representation of the example data about Peter Smith. The HTML 172  
attribute @about indicates that the current HTML element describes the entity 173  
identified by the URI reference <http://example.com/Peter>. The HTML attribute 174  
@rel specifies a relationship property between the anchor HTML element and the 175  
target URL. In the example, it uses the property foaf:knows to state that Peter 176  
knows Paula. For string properties, the attribute @property is introduced, as shown 177  
by usage of foaf:name to express Peter’s name. 178

A central idea of RDFa is the support for multiple, decentralized, independent, 179  
extensible vocabularies, in contrast to the community-driven centralized manage- 180  
ment of microformats. 181

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:foaf="http://xmlns.com/foaf/0.1/">
...
<div about="http://example.com/Peter" typeof="foaf:Person">
  <span property="foaf:name">Peter Smith</span> knows
  <a rel="foaf:knows" href="http://example.com/Paula">
    Paula Jones </a>.
</div>
...
```

**Fig. 1.3** RDFa representation of the example entity description

RDFa annotations are encouraged to directly reuse existing RDF vocabularies. 182  
However, for cases where no suitable vocabulary exists, data providers can define 183  
their own vocabularies, which in turn can be reused by third parties. Since anybody 184  
can define their own vocabularies, the possibility of naming conflicts raises an 185  
important concern as vocabularies get deployed in Web scale. This issue is handled 186  
in RDFa through the use of URI references for uniquely identifying vocabulary 187  
terms. Each vocabulary is identified by a URI that represents a namespace, and 188  
classes and properties within this vocabulary are prefixed with the vocabulary's 189  
namespace. For example, the Friend-of-a-Friend (FOAF) vocabulary is identified 190  
by the namespace <http://xmlns.com/foaf/0.1/>, and the property "homepage" 191  
is identified by <http://xmlns.com/foaf/0.1/homepage>. If somebody else has 192  
another property called "homepage" whose definition is incompatible with FOAF, 193  
they can place it in their own namespace (e.g., <http://example.com/homepage>). 194

As we have seen in the example, RDFa also supports the usage of URI references 195  
to identify real-world entities, such as Peter Smith, and provides for representing 196  
relationships between URI-identified entities (Peter and Paula). As URI references 197  
are globally unique, it is also possible to represent cross-server references and thus 198  
represent RDF links (see Sect. 1.5 on Linked Data) using RDFa. 199

### 1.3.2 *RDFa Deployment on the Web*

200

In March 2010, one of the major electronics retailers in the USA, BestBuy, 201  
started to publish 450,000 item descriptions using RDFa and the Good Relations 202  
vocabulary [27] for representing product data [36]. 203

In April 2010, Facebook announced<sup>11</sup> the Open Graph Protocol (<http://ogp.me/>), 204  
an RDFa-based markup for embedding structured data about people, places, 205  
organizations, products, films, as well as other entity types into HTML pages. 206

---

<sup>11</sup><http://developers.facebook.com/blog/post/377/>

**Table 1.1** Number of sites (second-level domains) publishing RDFa data divided by vocabulary

Vocabulary	Number of sites	
Dublin Core (dc)	344,545	t2.1
Open Graph Protocol (ogp)	177,761	t2.2
Creative Commons (cc)	37,890	t2.3
Google's Rich Snippets Vocabulary (rich)	6,083	t2.4
Friend-of-a-Friend (foaf)	2,545	t2.5
Semantically-Interlinked Online Communities (sioc)	1,633	t2.6
Electronic Business Cards (vcard)	1,349	t2.7
Good Relations (goodrel)	488	t2.8
Reviews (rev)	369	t2.9
CommonTag (tag)	272	t2.10
iCalendar Schema (ical)	62	t2.11
		t2.12

Whenever a Facebook user clicks the *Like* button on a page that is marked up using the Open Graph Protocol, Facebook imports the structured data from this page and uses it to describe the entity within other internal Facebook pages. Initial publishers using Open Graph Protocol included IMDB, Microsoft, NHL, Posterous, Rotten Tomatoes, TIME, and Yelp.<sup>12</sup>

In October 2010, another major online retailer, Overstock.com, announced the addition of RDFa to nearly one million pages.<sup>13</sup> Other prominent publishers of RDF include MySpace, BBC music, and Newsweek.

The RDFa usage statistics published by Yahoo! Research show that the usage of RDFa has increased between March 2009 and October 2010 from 72 to 430 million Web pages (a 510% increase) [34].

In June 2011, the W3C collected data<sup>14</sup> about the most used RDFa vocabularies in order to develop a default profile for RDFa 1.1.<sup>15</sup> Table 1.1 shows the number of sites (second-level domains) using each vocabulary. The numbers are again based on a crawl from Yahoo!.

The Dublin Core vocabulary is generally used on the Web to describe authorship of Web pages, and therefore is the most common vocabulary. Facebook's Open Graph Protocol has appeared in the second position, confirming that the company's endorsement of RDFa has been one of the factors in the rise of adoption. Licensing information through Creative Commons markup is the third most common use of RDFa, according to these stats. Product information, reviews, and offers are other prominent types of information in this sample and are represented using the Google's Rich Snippets, Good Relations, and Reviews vocabularies. Social

<sup>12</sup><http://rdfa.info/2010/04/22/facebook-adopts-rdfa/>

<sup>13</sup><http://rdfa.info/2010/10/07/overstock-com-adds-rdfa-to-nearly-one-million-pages/>

<sup>14</sup><http://www.w3.org/2010/02/rdfa/profile/data/>

<sup>15</sup><http://www.w3.org/profile/rdfa-1.1>

networking information is also well represented in the sample through SIOC and 230  
FOAF. 231

## 1.4 Microdata

232

Microdata<sup>16</sup> is an alternative proposal for embedding structured data into Web 233  
pages which was initially presented as part of the HTML 5 standardization effort 234  
in 2009. The development of HTML5 has been led by the WHATWG, a community 235  
of individuals from corporations such as Apple, Opera Software, and Mozilla 236  
Foundation whose interest is to evolve HTML and related technologies. Since 237  
May 2007, the W3C HTML Working Group decided to take WHATWG's HTML5 238  
specification as a starting point for the next major version of HTML. Microdata is 239  
an attempt to provide a simpler alternative to RDFa and microformats. It defines five 240  
new HTML attributes (as compared to zero for microformats and eight for RDFa), 241  
provides a unified syntax (in contrast to microformats), and allows for the usage of 242  
any vocabularies (similarly to RDFa). 243

The proposal has gained substantial attention since the announcement by Google, 244  
AQ10 Microsoft, and Yahoo! that they are collaborating on *Schema.org*, a collection of 245  
vocabularies for marking up content on Web pages [15]. Although the *Schema.org* 246  
vocabularies can be used in either RDFa or microdata markup, the initiative has 247  
chosen microdata as its preferred serialization syntax.<sup>17</sup> 248

Concerns were raised about the fact that the W3C currently has two draft 249  
specifications (microdata and RDFa) with the same objective, which prompted a 250  
decision from the W3C to separate the microdata from the HTML5 specification.<sup>18</sup> 251

### 1.4.1 Microdata Syntax

252

Figure 1.4 shows how our example data about Peter Smith are represented using 253  
*Schema.org* vocabulary terms and encoded using the microdata syntax. 254

Microdata consists of a group of name–value pairs. The groups are called items, 255  
and each name–value pair is a property. In order to mark up an item, the `itemscope` 256  
attribute is applied to an HTML element (line 1). The `itemid` attribute allows a 257  
Microdata item to be associated with a unique identifier (line 1). To add a property to 258  
an item, the `itemprop` attribute is used on one of the item's descendants (lines 2–3). 259

Microdata items can also be described with an entity type—e.g., on line 260  
1 <http://schema.org/Person>—which helps consumers to make sense and 261

<sup>16</sup><http://www.whatwg.org/specs/web-apps/current-work/multipage/microdata.html>

<sup>17</sup><http://schema.org/docs/gs.html>

<sup>18</sup><http://lists.w3.org/Archives/Public/public-html/2010Jan/att-0218/issue-76-decision.html>

```
<div itemscope itemtype="http://schema.org/Person"
      itemid="http://example.com/Peter.html">
  <span itemprop="name">Peter Smith</span>
  <a href="http://example.com/Paula.html"
      itemprop="knows">Paula Jones </a>
</div>
```

**Fig. 1.4** Microdata representation of the example entity description

properly display information. Types are identified using URI references. The item type is given as the value of an `itemtype` attribute on the same element as the `itemscope` attribute.

The value of an item property can itself be another item with its own set of properties. For example, Peter Smith knows a Paula Jones, who is also of type Person (line 3). In order to specify that the value of a property is another item, a new `itemscope` has to begin immediately after the corresponding `itemprop`.

Items can use nondescendant properties via the attribute `itemref`. This attribute is a list of the `ids` of properties or nested items elsewhere on the page.

Tool support for microdata is currently not yet widespread. No stable release of a Web browser supports the microdata DOM API. However, the microdataJS Javascript library can be used to emulate its behavior. However, the support for microdata is expected to rise quickly. The upcoming Opera 12 browser has announced support for microdata. Moreover, Google provides a Rich Snippets testing tool which can be used for validating microdata markup.

#### 1.4.2 Microdata Deployment on the Web

277

The *Schema.org* vocabularies cover over 220 different types of entities including creative works, events, organizations, persons, places, products, offers, and reviews.

For now, *Schema.org* is not aimed at covering the whole variety and richness of structured data on the Web. Therefore, it offers an extension mechanism that lets Web developers extend the existing vocabularies. By extending *Schema.org* schemas and using extensions to mark up data, search applications can at least partially understand the markup and use the data appropriately. Also, if the extended schema gains adoption and proves useful to search applications, search engines may start using these data. Extensions that gain significant adoption on the Web may be moved into the core *Schema.org* vocabulary.

As microdata is still in its early days, it is difficult to get solid statistics about microdata deployment. One source of statistics is the Sindice search engine which has crawled 329.13 million documents and thus operates on a much smaller corpus than the 12-billion-page corpus used for generating the Yahoo! statistics. As of July 2011, Sindice has crawled 2,085,050 documents containing microdata and using terms from the *Schema.org* vocabularies. The pages originate from around 100 Web sites. Table 1.2 shows the amount of entities for the ten most popular *Schema.org*

**Table 1.2** Schema.org type usage according to Sindice

Vocabulary term	Number of documents	
<a href="http://schema.org/Product">http://schema.org/Product</a>	1,253,153	t3.1
<a href="http://schema.org/Book">http://schema.org/Book</a>	264,918	t3.2
<a href="http://schema.org/Place">http://schema.org/Place</a>	201,622	t3.3
<a href="http://schema.org/Person">http://schema.org/Person</a>	191,078	t3.4
<a href="http://schema.org/Offer">http://schema.org/Offer</a>	136,630	t3.5
<a href="http://schema.org/Organization">http://schema.org/Organization</a>	17,980	t3.6
<a href="http://schema.org/LocalBusiness">http://schema.org/LocalBusiness</a>	13,300	t3.7
<a href="http://schema.org/Recipe">http://schema.org/Recipe</a>	4,815	t3.8
<a href="http://schema.org/Review">http://schema.org/Review</a>	3,519	t3.9
<a href="http://schema.org/Event">http://schema.org/Event</a>	442	t3.10
		t3.11

entity types as currently indexed by Sindice. Due to the recent interest in microdata 295  
following the announcement of *Schema.org*, these numbers are expected to rise 296  
significantly. 297

## 1.5 Linked Data

298

In contrast to microformats, RDFa, and Microdata, which provide for embedding 299  
structured data into HTML pages, the term *Linked Data* refers to a set of best 300  
practices for publishing structured data directly on the Web [10, 25]. These best 301  
practices were introduced by Tim Berners-Lee in his Web architecture note *Linked 302  
Data* [4] and have become known as the *Linked Data principles*. These principles 303  
are the following: 304

1. Use URIs as names for things. 305
2. Use HTTP URIs, so that people can look up those names. 306
3. When someone looks up a URI, provide useful information, using recommended 307  
standards (RDF, SPARQL). 308
4. Include links to other URIs, so that they can discover more things. 309

The first Linked Data principle advocates using URI references to identify not 310  
just Web documents and digital content but also real-world objects and abstract 311  
concepts. These may include tangible things, such as people, places, and cars, or 312  
those that are more abstract, such as the relationship type of *knowing somebody*, the 313  
set of all green cars in the world, or the color green itself. This principle can be seen 314  
as extending the scope of the Web from online resources to encompass any object 315  
or concept in the world. 316

The HTTP protocol is the Web's universal access mechanism. In the classic Web, 317  
HTTP URIs are used to combine globally unique identification with a simple, well- 318  
understood retrieval mechanism. Thus, the second Linked Data principle advocates 319  
the use of HTTP URIs to identify objects and abstract concepts, enabling these URIs 320  
to be *dereferenced* (i.e., looked up) over the HTTP protocol into a description of the 321  
identified object or concept. 322

In order to enable a wide range of different applications to process Web content, 323  
it is important to agree on standardized content formats. The agreement on HTML 324  
as a dominant document format was an important factor that made the Web scale. 325  
The third Linked Data principle therefore advocates use of a single data model for 326  
publishing structured data on the Web—the resource description framework (RDF), 327  
a simple graph-based data model that has been designed for use in the context of the 328  
Web [31]. For serializing RDF data, the RDF/XML syntax [2] is widely used in the 329  
Linked Data context. 330

The fourth Linked Data principle advocates the use of hyperlinks to connect not 331  
only Web documents but also any type of thing. For example, a hyperlink may be 332  
set between a person and a place or between a place and a company. In contrast 333  
to the classic Web where hyperlinks are largely untyped, hyperlinks that connect 334  
things in a Linked Data context have types which describe the relationship between 335  
the things. For example, a hyperlink of the type *friend of* may be set between two 336  
people, or a hyperlink of the type *based near* may be set between a person and 337  
a place. Hyperlinks in the Linked Data context are called *RDF links* in order to 338  
distinguish them from hyperlinks between classic Web documents. 339

Across the Web, many different servers are responsible for answering requests 340  
attempting to dereference HTTP URIs in many different namespaces and (in a 341  
Linked Data context) returning RDF descriptions of the resources identified by these 342  
URIs. Therefore, in a Linked Data context, if an RDF link connects URIs in different 343  
namespaces, it ultimately connects resources in different datasets. 344

Just as hyperlinks in the classic Web connect documents into a single global 345  
information space, Linked Data uses hyperlinks to connect disparate data into a 346  
single global dataspace. These links, in turn, enable applications to navigate the 347  
dataspace. For example, a Linked Data application that has looked up a URI and 348  
retrieved RDF data describing a person may follow links from those data to data on 349  
different Web servers, describing, for instance, the place where the person lives or 350  
the company for which the person works. 351

These Linked Data principles enable the implementation of generic applications 352  
that operate over the complete dataspace, since the resulting Web of Linked Data is 353  
based on standards for the identification of entities, retrieval of entity descriptions, 354  
and parsing of descriptions in RDF as common data model. Examples of such 355  
applications include Linked Data browsers which enable the user to view data 356  
from one data source and then follow RDF links within the data to other data 357  
sources. Other examples are Linked Data search engines, such as *Sig.ma*<sup>19</sup> [41], 358  
*Falcons*<sup>20</sup> [21], *SWSE*<sup>21</sup> [24], and *VisiNav*<sup>22</sup> [22], which crawl the Web of Linked 359  
Data and provide advanced query capabilities on top of the dataspace. 360

<sup>19</sup><http://sig.ma/>

<sup>20</sup><http://iws.seu.edu.cn/services/falcons/documentsearch/>

<sup>21</sup><http://www.swse.org/>

<sup>22</sup><http://sw.deri.org/2009/01/visinav/>

Linked Data builds directly on the Web architecture [29] and applies this 361 architecture to the task of publishing and interconnecting data on global scale. 362 Linked Data thus strongly advocates the usage of URIs for the globally unique 363 identification of real-world entities. This practice distinguishes Linked Data from 364 microformats, RDFa, and microdata which also provide language features for 365 the globally unique identification of entities but where these features are less 366 widely used in practice. Another difference between Linked Data and the other 367 formats is the strong focus of Linked Data on setting RDF links that connect 368 data between different servers. Setting links between entities is also possible using 369 the RDFa and microdata syntaxes, but the focus of these technologies has been 370 to describe entities within one Web page, rather than interconnecting disparate 371 sources. 372

### 1.5.1 RDF/XML Syntax

373

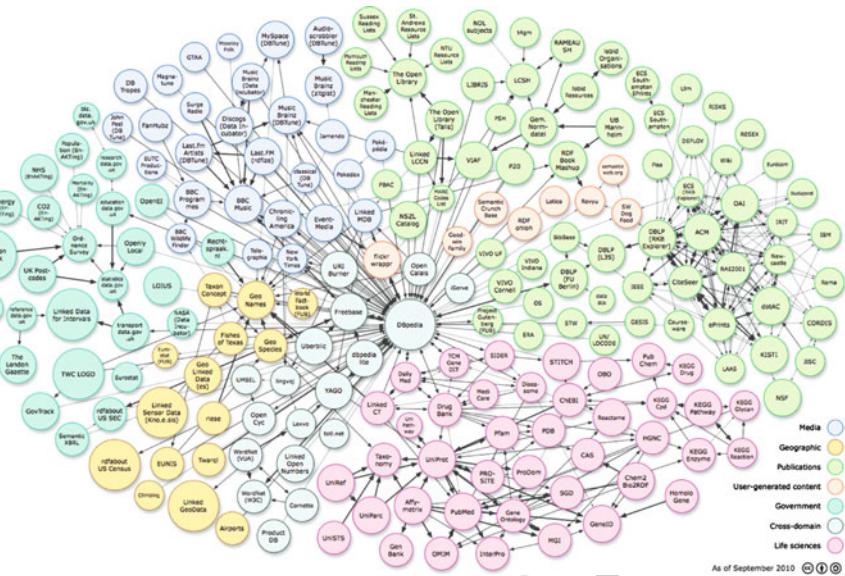
Whenever a Linked Data client looks up an HTTP URI over the HTTP protocol 374 and asks for the content type application/rdf+xml, the corresponding Web 375 server returns an RDF description of the identified object using the RDF/XML 376 syntax [2]. 377

Figure 1.5 shows how our example description of Peter Smith is serialized using 378 the RDF/XML syntax. Lines 2 and 3 define the namespaces of the vocabularies 379 used within the description. It is interesting to note that it is common practice in the 380 Linked Data context to mix terms from different widely used vocabularies to cover 381 different aspects of the description. In our case, we use the RDF base vocabulary as 382 well as FOAF, a vocabulary for describing people. Lines 6 and 7 state that there is a 383 thing, identified by the URI <http://example.com/People/Peter> (line 6) of type 384 foaf:Person (line 7). Line 8 states that this thing has the name Peter Smith. Line 9 385 states that Peter Smith knows Paula Jones, which is identified by the URI reference 386 <http://example.com/People/Paula>. 387

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/">

    <rdf:Description rdf:about="http://example.com/People/Peter">
        <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
        <foaf:name>Peter Smith </foaf:name>
        <foaf:knows rdf:resource="http://example.com/People/Paula"/>
    </rdf:Description>
</rdf:RDF>
```

Fig. 1.5 RDF/XML representation of the example entity description about Peter Smith



**Fig. 1.6** Linking Open Data cloud as of November 2010. The colors classify datasets by topical domain

### 1.5.2 Linked Data Deployment on the Web

388

The deployment of Linked Data on the Web was initialized by the W3C *Linking Open Data (LOD) project*,<sup>23</sup> a grassroots community effort founded in January 2007. The founding aim of the project was to identify existing datasets that are available under open licenses, convert them to RDF according to the Linked Data principles, and publish them on the Web.

Figure 1.6 illustrates the November 2010 scale of the *Linked Data cloud* originating from the W3C Linking Open Data project and classifies the datasets by topical domain, highlighting the diversity of datasets present in the cloud. Each node in the diagram represents a distinct dataset published as Linked Data. The arcs indicate that RDF links exist between items in the two connected datasets. Heavier edges roughly correspond to a greater number of links between two datasets, and bidirectional arcs indicate the existence of outward links to one another.

The LOD community maintains a catalog of all known Linked Data sources, the *LOD Cloud Data Catalog*.<sup>24</sup> Table 1.3 gives an overview of the data sources that are cataloged as of July 2011. For each topical domain, the table lists the number of data sources, the number of RDF triples served by these data sources, as well as the number of RDF links that are set between data sources. The numbers are taken

<sup>23</sup><http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData>

<sup>24</sup><http://www.ckan.net/group/lodcloud>

**Table 1.3** Number of datasets, amount of triples, and amount of RDF links per topical domain

Domain	Datasets	Triples	Percent	RDF links	Percent	t4.1
Cross-domain	36	3,904,150,829	12.59	48,616,441	10.42	t4.2
Geographic	23	6,002,838,754	19.36	35,750,600	7.66	t4.3
Government	40	13,230,112,776	42.67	18,581,087	3.98	t4.4
Media	27	1,855,413,060	5.98	50,374,304	10.82	t4.5
Libraries	83	2,866,021,476	9.24	112,233,089	24.05	t4.6
Life sciences	43	3,034,893,765	9.79	197,576,686	42.34	t4.7
User Content	13	114,203,457	0.37	3,423,090	0.73	t4.8
	265	31,007,634,117		466,671,476		t4.9

from the *State of the LOD Cloud* document<sup>25</sup> which on a regular basis compiles 406  
summary statistics about the datasets that are cataloged within the LOD Cloud Data 407  
Catalog. 408

Altogether, the 265 data sources that are currently cataloged serve over 31 billion 409  
RDF triples to the Web. A total of 466 million of these triples are RDF links that 410  
connect entity descriptions from different data sources. 411

In the following sections, we will give a short overview of major datasets from 412  
each topical domain and highlight the wide variety of the available data. 413

### 1.5.2.1 Cross-Domain Data

Some of the first datasets that appeared in the Web of Linked Data are not 415  
specific to one topic, but span multiple domains. This cross-domain coverage is 416  
crucial for helping to connect domain-specific datasets into a single, interconnected 417  
dataspace, thereby avoiding fragmentation of the Web of Data into isolated, topical 418  
*data islands*. Cross-domain datasets often serve as linkage hubs on the Web of 419  
Data. 420

The prototypical example of cross-domain Linked Data is *DBpedia*<sup>26</sup> [13], a 421  
Linked Data version of Wikipedia. Things that are the subject of a Wikipedia 422  
article are automatically assigned a DBpedia URI, based on the URI of that 423  
Wikipedia article. For example, the Wikipedia article about the city of Berlin has the 424  
following URI <http://en.wikipedia.org/wiki/Berlin>. Therefore, Berlin has 425  
the corresponding DBpedia URI <http://dbpedia.org/resource/Berlin>, which 426  
is not the URI of a Web page about Berlin, but a URI that identifies the city 427  
itself. RDF statements that refer to this URI are then generated by extracting 428  
information from various parts of the Wikipedia articles, in particular the *infoboxes* 429  
commonly seen on the right-hand side of Wikipedia articles. Because of its breadth 430  
of topical coverage, DBpedia has served as a hub within the Web of Linked Data 431

<sup>25</sup><http://lod-cloud.net/state/>

<sup>26</sup><http://dbpedia.org/>

from the early stages of the Linking Open Data project. The wealth of inward and outward links connecting items in DBpedia to items in other datasets is apparent in Fig. 1.6. 432  
433  
434

A second major source of cross-domain Linked Data is *Freebase*,<sup>27</sup> an editable, 435  
openly licensed database populated through user contributions and data imports 436  
from sources such as Wikipedia and Geonames. Freebase provides RDF descrip- 437  
tions of items in the database, which are linked to items in DBpedia with incoming 438  
and outgoing links. 439

Further cross-domain datasets include UMBEL,<sup>28</sup> YAGO [40], and OpenCyc.<sup>29</sup> 440  
These are, in turn, linked with DBpedia, helping to facilitate data integration across 441  
a wide range of interlinked sources. 442

Besides a rich pool of entity data, the cross-domain datasets contain a lot of 443  
domain knowledge in the form of taxonomic structures which makes them a rich 444  
source of background knowledge for search applications. 445

### 1.5.2.2 Geographic Data

446

Geographical datasets can often connect information from varied topical domains. 447  
This is apparent in the Web of Data, where the *Geonames*<sup>30</sup> dataset frequently serves 448  
as a hub for other datasets that cover geographical data, also only partly. Geonames 449  
is an open-license geographical database that publishes Linked Data about 8 million 450  
locations. 451

A second significant dataset in this area is *LinkedGeoData* [39], a Linked Data 452  
conversion of data from the *OpenStreetMap* project, which provided information 453  
about more than 350 million spatial features. Wherever possible, locations in 454  
Geonames and LinkedGeoData are interlinked with corresponding locations in 455  
DBpedia, ensuring there is a core of interlinked data about geographical locations. 456

Linked Data versions of the EuroStat,<sup>31</sup> World Factbook,<sup>32</sup> and US Census<sup>33</sup> 457  
datasets begin to bridge the worlds of statistics, politics, and social geography, 458  
while *Ordnance Survey* (the national mapping agency of Great Britain) has begun to 459  
publish Linked Data describing the administrative areas within the Great Britain,<sup>34</sup> 460  
in efforts related to the *data.gov.uk* initiative are described in Sect. 1.5.2.4. 461

AQ11

<sup>27</sup><http://www.freebase.com>

<sup>28</sup><http://umbel.org/>

<sup>29</sup><http://sw.opencyc.org/>

<sup>30</sup><http://www.geonames.org/>

<sup>31</sup><http://ckan.net/package?q=eurostat&groups=lodcloud>

<sup>32</sup><http://www4.wiwiiss.fu-berlin.de/factbook/>

<sup>33</sup><http://www.rdfabout.com/demo/census/>

<sup>34</sup><http://data.ordnancesurvey.co.uk/>

**1.5.2.3 Media Data**

462

One of the first large organizations to recognize the potential of Linked Data and 463  
adopt the principles and technologies into their publishing and content management 464  
workflows has been the British Broadcasting Corporation (BBC). Following earlier 465  
experiments with publishing their catalogue of programs as RDF, the BBC released 466  
in 2008 two large sites that combine publication of Linked Data and conventional 467  
Web pages. The first of these, */programmes*,<sup>35</sup> provides a URI for and RDF 468  
description of every episode of the TV or radio program broadcast across the BBC's 469  
various channels [32]. 470

The second of these sites, */music*,<sup>36</sup> publishes Linked Data about every artist 471  
whose music has been played on BBC radio stations. This type of music data 472  
is interlinked with DBpedia, and it receives incoming links from a range of 473  
music-related Linked Data sources. These cross dataset links allow applications to 474  
consume data from all these sources and integrate them to provide rich artist profiles, 475  
while the playlist data can be mined to find similarities between artists that may be 476  
used to generate recommendations. 477

A second major publisher of Linked Data in the media industry is the New York 478  
Times. The New York Times has published a significant proportion of its internal 479  
subject headings as Linked Data<sup>37</sup> under a *Creative Commons Attribution* license, 480  
interlinking these topics with DBpedia, Freebase, and Geonames. The intention is 481  
to use these liberally licensed data as a map to lead people to the rich archive of 482  
content maintained by the New York Times. 483

**1.5.2.4 Government Data**

484

Governmental bodies and public-sector organizations produce a wealth of data, 485  
ranging from economic statistics, to registers of companies and land ownership, 486  
reports on the performance of schools, crime statistics, and the voting records of 487  
elected representatives. 488

The potential of Linked Data for easing the access to government data is 489  
increasingly understood, with both the *data.gov.uk*<sup>38</sup> and *data.gov*<sup>39</sup> initiatives 490  
publishing significant volumes of data on the Web according to the Linked Data 491  
principles. The approach taken in the two countries differs slightly: to date, the 492  
latter has converted very large volumes of data, while the former has focused on the 493  
creation of core data-level infrastructure for publishing Linked Data, such as stable 494  
URIs to which increasing amounts of data can be connected [38]. 495

<sup>35</sup><http://www.bbc.co.uk/programmes>

<sup>36</sup><http://www.bbc.co.uk/music>

<sup>37</sup><http://data.nytimes.com/>

<sup>38</sup><http://data.gov.uk/linked-data>

<sup>39</sup><http://www.data.gov/semantic>

A very interesting initiative is being pursued by the *UK Civil Service*,<sup>40</sup> which has started to mark up job vacancies using RDFa. By providing information about open positions in a structured form, it becomes easier for external job portals to incorporate civil service jobs [8]. If more organizations would follow this example, the transparency in the labor market could be significantly increased [11].

In order to provide a forum for coordinating the work on using Linked Data and other Web standards to improve access to government data and increase government transparency, W3C has formed an eGovernment Interest Group.<sup>41</sup>

### 1.5.2.5 Libraries and Education

With an imperative to support novel means of discovery and a wealth of experience in producing high-quality structured data, libraries are natural complementors to Linked Data. This field has seen some significant early developments which aim at integrating library catalogs on a global scale; interlinking the content of multiple library catalogs, for instance, by topic, location, or historical period; interlink library catalogs with third-party information (picture and video archives, or knowledge bases like DBpedia); and at making library data more easily accessible by relying on Web standards.

Examples include the American Library of Congress and the German National Library of Economics which publish their subject heading taxonomies as Linked Data (see Note<sup>42</sup> and [37], respectively), while the complete content of LIBRIS and the Swedish National Union Catalogue is available as Linked Data.<sup>43</sup> Similarly, the *OpenLibrary*, a collaborative effort to create “one Web page for every book ever published,”<sup>44</sup> publishes its catalogue in RDF.

Scholarly articles from journals and conferences are also well represented in the Web of Data through community publishing efforts such as DBLP as Linked Data,<sup>45</sup> RKBexplorer,<sup>46</sup> and the Semantic Web Dog food Server<sup>47</sup> [35].

High levels of ongoing activity in the library community will no doubt lead to further significant Linked Data deployments in this area. Of particular note in this area is the new *Object Reuse and Exchange (OAI-ORE)* standard from the *Open Archives Initiative* [42], which is based on the Linked Data principles. The OAI-ORE, Dublin Core, SKOS, and FOAF vocabularies form the foundation

<sup>40</sup><http://www.civilservice.gov.uk/>

<sup>41</sup>[http://www.w3.org/egov/wiki/Main\\\_\\\_Page](http://www.w3.org/egov/wiki/Main\_\_Page)

<sup>42</sup><http://id.loc.gov/authorities/about.html>

<sup>43</sup><http://blog.libris.kb.se/semweb/?p=7>

<sup>44</sup><http://openlibrary.org/>

<sup>45</sup><http://dblp.l3s.de/>; <http://www4.wiwiiss.fu-berlin.de/dblp/>; <http://dblp.rkbexplorer.com/>

<sup>46</sup><http://www.rkbexplorer.com/data/>

<sup>47</sup><http://data.semanticweb.org/>

of the new Europeana Data Model.<sup>48</sup> The adoption of this model by libraries, museums, and cultural institutions that participate in Europeana will further accelerate the availability of Linked Data related to publications and cultural heritage artifacts.

In order to provide a forum and to coordinate the efforts to increase the global interoperability of library data, W3C has started a Library Linked Data Incubator Group.<sup>49</sup>

### 1.5.2.6 Life Sciences Data

Linked Data has gained significant uptake in the life sciences as a technology to connect the various datasets that are used by researchers in this field. In particular, the Bio2RDF project [3] has interlinked more than 30 widely used datasets, including UniProt (the Universal Protein Resource), KEGG (the Kyoto Encyclopedia of Genes and Genomes), CAS (the Chemical Abstracts Service), PubMed, and the Gene Ontology. The W3C *Linking Open Drug Data* effort<sup>50</sup> has brought together the pharmaceutical companies Eli Lilly, AstraZeneca, and Johnson & Johnson, in a cooperative effort to interlink openly licensed data about drugs and clinical trials, in order to aid drug discovery [30].

### 1.5.2.7 User-Generated Content and Social Media

Some of the earliest datasets in the Web of Linked Data were based on conversions of, or wrappers around, *Web 2.0* sites with large volumes of *user-generated content*. This has produced datasets and services such as the *FlickrWrappr*,<sup>51</sup> a Linked Data wrapper around the Flickr photo-sharing service. These were complemented by user-generated content sites that were built with native support for Linked Data, such as *Revyu.com* [26] for reviews and ratings, and Faviki<sup>52</sup> for annotating Web content with Linked Data URIs.

Initially, the Linked Data best practices were adopted mainly by research projects and Web enthusiasts. These third parties took existing datasets, converted them into RDF, and served them on the Web. Alternatively, they implemented Linked Data wrappers around existing Web APIs. Today, Linked Data technologies are increasingly adopted by the primary data producers themselves and are used by

<sup>48</sup>[http://version1.europeana.eu/c/document/\\_library/get/\\_file?uuid=9783319c-9049-436c-bdf9-25f72e85e34c&groupId=10602](http://version1.europeana.eu/c/document/_library/get/_file?uuid=9783319c-9049-436c-bdf9-25f72e85e34c&groupId=10602)

<sup>49</sup><http://www.w3.org/2005/Incubator/lld>

<sup>50</sup><http://esw.w3.org/HCLSIG/LODD>

<sup>51</sup><http://www4.wiwiiss.fu-berlin.de/flickrwrappr/>

<sup>52</sup><http://www.faviki.com/>

them to provide access to their datasets. As of July 2011, out of the 265 datasets 557 in the LOD cloud, 100 (37.88%) are published by the data producers themselves, 558 while 164 (62.12%) are published by third parties. 559

### 1.5.2.8 Vocabulary Usage by Linked Data Sources

560

A common practice in the Linked Data community is to reuse terms from widely 561 deployed vocabularies, whenever possible, in order to increase homogeneity of 562 descriptions and, consequently, ease the understanding of these descriptions. As 563 Linked Data sources cover a wide variety of topics, widely deployed vocabularies 564 that cover all aspects of these topics may not exist yet. Therefore, publishers 565 commonly also define proprietary terms and mix these terms with the widely used 566 ones in order to cover the more specific aspects and to publish the complete content 567 of a dataset on the Web. 568

Nearly all data sources in the LOD cloud use terms from the W3C base 569 vocabularies RDF, RDF Schema, and OWL. In addition, 174 (65.66%) of the 265 570 data sources in the LOD cloud catalog use terms from other widely deployed 571 vocabularies. Table 1.4 shows the distribution of the most widely used vocabularies 572 in the Linked Data context. 573

It is interesting to note that these widely used vocabularies cover generic types of 574 entities (such as the Friend-of-a-Friend (FOAF) vocabulary<sup>53</sup> being used to describe 575 people or the Basic Geo Vocabulary (geo)<sup>54</sup> being used to describe locations) 576 but that domain-specific vocabularies such as the Music Ontology (mo)<sup>55</sup> or the 577 Bibliographic Ontology (bibo)<sup>56</sup> are also starting to gain traction. 578

**Table 1.4** Number of linked data sources using specific vocabularies

Vocabulary	Number of data sources	
Dublin Core(dc)	162 (61.13%)	t5.1
Friend-of-a-Friend (foaf)	73 (27.55%)	t5.2
Simple Knowledge Organization System (skos)	46 (17.36%)	t5.3
Basic Geo Vocabulary (geo)	21 (7.92%)	t5.4
AKTive Portal (akt)	17 (6.42%)	t5.5
Music Ontology (mo)	13 (4.91%)	t5.6
Bibliographic Ontology (bibo)	11 (4.15%)	t5.7
Semantically-Interlinked Online Communities (sioc)	8 (3.02%)	t5.8
Creative Commons (cc)	7 (2.64%)	t5.9
Electronic Business Cards (vcard)	7 (2.64%)	t5.10
		t5.11

<sup>53</sup><http://xmlns.com/foaf/0.1/>

<sup>54</sup>[http://www.w3.org/2003/01/geo/wgs84\\\_pos](http://www.w3.org/2003/01/geo/wgs84\_pos)

<sup>55</sup><http://purl.org/ontology/mo/>

<sup>56</sup><http://bibliontology.com/>

A total of 171 (64.53%) out of the 265 data sources in the LOD cloud use 579 proprietary vocabulary terms in addition to widely used terms, while 93 (35.09%) 580 do not use proprietary vocabulary terms. In order to enable applications to automatically 581 retrieve the definition of vocabulary terms from the Web, URIs identifying 582 vocabulary terms should be made dereferenceable. Guidelines for doing this are 583 given in the W3C note Best Practice Recipes for Publishing RDF Vocabularies.<sup>57</sup> 584 This best practice is currently implemented by 148 (86.55%) out of these 171 data 585 sources that use proprietary term, while 23 (13.45%) do not make proprietary term 586 URIs dereferenceable. 587

A central idea of Linked Data is to set RDF links between entities on the 588 Web. This does not only apply to instance data but can also be applied to 589 publish correspondences between vocabulary terms on the Web and thus provide 590 integration hints that enable Linked Data applications to translate data between 591 different vocabularies. The W3C recommendations define the following terms 592 for representing such correspondences (mappings): owl:equivalentClass, 593 owl:equivalentProperty, or if a looser mapping is desired: rdfs:subClassOf, 594 rdfs:subPropertyOf, and skos:broadMatch, skos:narrowMatch. 595

The Web site Linked Open Vocabularies<sup>58</sup> tracks the RDF links that are set 596 between terms from 127 nonproprietary vocabularies that are used in the Linked 597 Data context. According to the CKAN.net dataset catalog, 13 (7.60%) out of the 598 171 data sources that use proprietary terms provide mappings to other vocabularies 599 for their terms. 600

## 1.6 Evaluation Data

601

As the deployment of structured data on the Web rises, the opportunities for 602 semantic search applications increase, and the need for evaluation datasets to test 603 and compare these applications becomes evident. One approach to obtain Web data 604 is to use publicly available software for crawling the Web, such as Nutch<sup>59</sup> for 605 crawling Web pages and LDSpider [28] for crawling Linked Data. However, there 606 exist already a number of publicly available evaluation datasets that have been 607 crawled from the Web and can be promptly used for evaluating semantic search 608 applications. 609

- *ClueWeb09*. The ClueWeb09 corpus is a collection of approximately 1 billion 610 Web pages (25TB) in ten languages[18]. The corpus has been crawled in January 611 and February 2009 and contains microdata as well as RDFa data. The corpus has 612 been extensively used in the Text Retrieval Conference (TREC) for evaluating 613 information retrieval and other related human language technologies. 614

<sup>57</sup><http://www.w3.org/TR/2008/NOTE-swbp-vocab-pub-20080828/>

<sup>58</sup><http://labs.mondeca.com/dataset/lov/index.html>

<sup>59</sup><http://nutch.apache.org/>

AQ13

- *TREC Entity*. One of the extensions of the ClueWeb09 corpus, which is of particular interest to semantic search applications, was developed for the TREC Entity Retrieval track. This conference track aims at evaluating the ability of systems to find related entities on Web pages. The TREC Entity dataset defines topics, which consist of information needs and acceptable answers. One example information need is “recording companies selling the Kingston Trio’s songs.” Acceptable answers for that information need would include home pages of the companies sought after—e.g., <http://www.capitolrecords.com/> and <http://www.deccarecords-us.com/>—retrieved from the ClueWeb09 collection. The 2010 edition of TREC Entity developed 50 of such topics, which were extended by other 50 topics in its 2011 edition.
- *Sindice 2011*. The Sindice 2011 dataset has been crawled from the Web by the Sindice search engine [19]. The dataset consists of 11 billion RDF triples which have been extracted from 230 million Web documents containing RDF, RDFa, and Microformats and have been crawled from the Web of Linked Data. The documents originate from 270,000 second-level domains. The dataset contains descriptions of about 1.7 billion entities. Detailed statistics are provided on the dataset page.<sup>60</sup> The Sindice 2011 dataset is used in the TREC Entity 2011 evaluation campaign, which introduced an LOD variant, where the retrieved entity identifiers are Linked Open Data URIs instead of homepage URLs, e.g., dbpedia:Sony\_BMG and dbpedia:Universal\_Music\_Group.
- *Billion Triple Challenge*. Precrawled datasets that only contain Linked Data are the Billion Triple Challenge (BTC) datasets dating from 2009, 2010, and 2011 [23]. The latest version, BTC2011, has been crawled in May/June 2011 and consists of 2 billion RDF triples from Linked Data sources. The BTC data are employed in the Semantic Web Challenge, an academic competition that happens as part of the International Semantic Web Conference. The central idea of this competition is to apply Semantic Web techniques for the construction of Web applications that integrate, combine, and create information needed to perform user-driven tasks.
- *SemSearch 2011*. Yet another prominent example use of the BTC datasets is the Semantic Search Challenge in the World Wide Web Conference. The Semantic Search Challenge establishes an academic competition for the best systems that can answer a number of queries that are focused on the task of entity search. Two sets of queries are provided. For the entity search track, the queries are a subset from the Yahoo! Search Query Tiny Sample, available from the Yahoo! Webscope.<sup>61</sup> For the list search track, the queries have been handwritten by the organizing committee. The evaluation of the results was performed through a crowdsourcing process[14] that used Amazon’s Mechanical Turk.<sup>62</sup>

<sup>60</sup><http://data.sindice.com/trec2011/statistics.html>

<sup>61</sup><http://webscope.sandbox.yahoo.com/>

<sup>62</sup><http://mturk.com>

## 1.7 Conclusion

654

This chapter gave an overview of the different techniques that are currently used to 655  
publish structured data on the Web as well as about the types and amounts of data 656  
that are currently published on the Web using each technique. The chapter has shown 657  
that the Web of Data is not a vision anymore, but it is actually there. For search 658  
engines, this means that they can draw on a large corpus of structured data that 659  
is embedded into HTML pages and helps them to understand the content of these 660  
pages. The Web of Data also contains many datasets which are directly published on 661  
the Web according to the Linked Data principles. These datasets provide a rich pool 662  
of detailed domain-specific entity data as well as taxonomic classifications which 663  
can both be used as background knowledge to improve search applications. 664

Looking at the types of data that are currently published on the Web, we can 665  
conclude that Microformats, RDFa, and Mircodata are mostly used to publish data 666  
that are directly relevant for end-user queries—such as data describing products, 667  
reviews, people, organizations, news, and events. The range of data that are 668  
published as Linked Data is wider and includes many very specific entity types such 669  
as statistical data, government data, research data, library data, as well as cross- 670  
domain datasets like DBpedia and Freebase. Another distinguishing factor between 671  
Microformats, RDFa, and Linked Data is the emphasis on setting links between 672  
entity descriptions. While Microformats and RDFa data mostly remain isolated and 673  
existing language features to connect entries between data sources are not widely 674  
used, the focus of the Linked Data community lies clearly on interconnecting data 675  
between different sources and thus eases the discovery as well as data integration. 676

Applications that want to exploit the Web of Data are facing two main challenges 677  
today: 678

- *Semantic Heterogeneity.* The different techniques that are used to publish data on 679  
the Web lead to a certain degree of syntax heterogeneity. But as current search 680  
engines like Google, Yahoo!, Bing, and Sindice show, this syntax heterogeneity 681  
can easily be bridged by using multiple parsers in parallel. What is harder to 682  
overcome is the semantic heterogeneity that results from different data sources 683  
using different vocabularies to represent data about the same types of entities. 684  
The semantic heterogeneity is further increased by the fact that, in an open 685  
environment like the Web, different data sources also use the same vocabulary 686  
in different ways, for instance, squeezing different types of values into the same 687  
property. A coarse-grained starting point to handle semantic heterogeneity is the 688  
increased adoption of common vocabularies by many data sources [6] as well 689  
as the practice to set equivalence and subclass/subproperty links between terms 690  
from different vocabularies that have emerged in the Linked Data commu- 691  
nity [25]. Advanced techniques to tackle semantic heterogeneity in the context 692  
of the Web include schema clustering [33] as well as machine-learning-based 693  
techniques that derive correspondences by looking at both—the vocabulary and 694  
the instance level [7]. Such methods are likely to produce good results in 695  
the context of the Web of Data as large amounts of instance data are readily 696

available on the Web for training. Besides these potentials, high-quality data integration will always require human intervention. The current challenge for search applications is to find the appropriate mix of both. 697  
698  
699

- *Data Quality.* The Web is an open medium and everybody can publish data on the Web. Thus, the Web will always contain data that are outdated, conflicting, or intensionally wrong (spam). As search applications start to aggregate data from growing numbers of Web data sources, the task of assessing the quality of Web data and deciding which data to trust becomes increasingly challenging. Groundwork for tracking the provenance of Web data is currently done by the W3C Provenance Working Group.<sup>63</sup> The range of data quality assessment methods that are potentially applicable in the Web context is very wide, and the right mixture of these methods will depend on the application context. Data quality assessment is likely to again involve human intervention, at least in the form of manually assessing explanations provided by a system about its trust decisions [9]. 700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711

Handling semantic heterogeneity and data quality in the Web context poses interesting research challenges. But as current search engines like Google, Sig.ma, Falcons show, pragmatic approaches already suffice a long way. 712  
713  
714

The better we learn to handle these challenges, the more sophisticated the functionality of search engines can get. The development of applications that exploit the Web of Data is just starting, and we are likely to see a wide range of innovative applications emerge around the topics of Web-scale data integration and aggregation as well as Web-scale data mining. 715  
716  
717  
718  
719

## References

720

- AQ15
1. Adida, B., Birbeck, M.: Rdfa primer—bridging the human and data webs—w3c recommendation. <http://www.w3.org/TR/xhtml-rdfa-primer/> (2008) URL <http://www.w3.org/TR/xhtml-rdfa-primer/> 721  
722
  2. Beckett, D.: RDF/XML Syntax Specification (Revised)—W3C Recommendation. <http://www.w3.org/TR/rdf-syntax-grammar/> (2004) 724  
725
  3. Belleau, F., Nolin, M., Tourigny, N., Rigault, P., Morissette, J.: Bio2rdf: Towards a mashup to build bioinformatics knowledge systems. *J. Biomed. Inform.* **41**(5), 706–716 (2008) 726  
727
  4. BernersLee, T.: Linked data design issues. <http://www.w3.org/DesignIssues/LinkedData.html> (2006) 728  
729
  5. Berners-Lee, T.: Giant global graph. <http://dig.csail.mit.edu/breadcrumbs/node/215> (2007). URL <http://dig.csail.mit.edu/breadcrumbs/node/215> 730  
731
  6. Berners-Lee, T., Kagal, L.: The fractal nature of the semantic web. *AI Magazine* **29**(3) (2008) 732
  7. Bilke, A., Naumann, F.: Schema matching using duplicates. Proceedings of the International Conference on Data Engineering, 05–08 April 2005 733  
734
  8. Birbeck, M.: Rdfa and linked data in uk government web-sites. *Nodalities Magazine* **7** (2009) 735

---

<sup>63</sup>[http://www.w3.org/2011/prov/wiki/Main\\_Page](http://www.w3.org/2011/prov/wiki/Main_Page)

9. Bizer, C., Cyganiak, R.: Quality-driven information filtering using the wiqa policy framework. *J. Web Semant. Sci. Serv. Agents World Wide Web* **7**(1), 1–10 (2009) 736  
737
10. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. *Int. J. Semant. Web Inf. Syst.* **5**(3), 1–22 (2009) 738  
739
11. Bizer, C., Heese, R., Mochol, M., Oldakowski, R., Tolksdorf, R., Eckstein, R.: The impact of semantic web technologies on job recruitment processes. *Proceedings of the 7th Internationale Tagung Wirtschaftsinformatik (WI2005)*, 2005 740  
741  
742
12. Bizer, C., Jentzsch, A., Cyganiak, R.: State of the lod cloud. <http://www4.wiwiiss.fu-berlin.de/lodcloud/state/> (2011). URL <http://www4.wiwiiss.fu-berlin.de/lodcloud/state/> 743  
744
13. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: Dbpedia – a crystallization point for the web of data. *J. Web Semant. Sci. Serv. Agents World Wide Web* **7**(3), 154–165 (2009) 745  
746  
747
14. Blanco, R., Halpin, H., Herzig, D.M., Mika, P., Pound, J., Thompson, H.S., Tran Duc, T.: Repeatable and reliable search system evaluation using crowdsourcing. *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information, SIGIR '11*, ACM, New York, NY, USA, 2011, pp. 923–932 DOI <http://doi.acm.org/10.1145/2009916.2010039> 748  
749  
750  
751  
752
15. Blog, G.: Introducing schema.org: search engines come together for a richer web. <http://googleblog.blogspot.com/2011/06/introducing-schemaorg-search-engines.html> (2011) 753  
754
16. Blog, Y.S.: We now support microformats. <http://www.ysearchblog.com/2006/06/21/we-now-support-microformats/> (2006) 755  
756
17. Blog, Y.S.: The yahoo! search open ecosystem. <http://www.ysearchblog.com/archives/000527.html> (2008) 757  
758
18. Callan, J.: The Sapphire Web Crawler – crawl statistics. <http://boston.lti.cs.cmu.edu/crawler/crawlerstats.html> (2009) 759  
760
19. Campinas, S., Ceccarelli, D., Perry, T.E., Delbru, R., Balog, K., Tummarello, G.: The Sindice-2011 dataset for entity-oriented search in the web of data. *1st international workshop on entity-oriented search (EOS) 2011* 761  
762  
763
20. Central, G.W.: Introducing rich snippets. <http://googlewebmastercentral.blogspot.com/2009/05/introducing-rich-snippets.html> (2011) 764  
765
21. Cheng, G., Qu, Y.: Searching linked objects with falcons: approach, implementation and evaluation. *Int. J. Semant. Web Inform. Syst. (IJSWIS)* **5**(3), 49–70 (2009) 766  
767
22. Harth, A.: Visinav: a system for visual search and navigation on web data. *Web Semant. Sci. Serv. Agents World Wide Web* **8**(4), 348–354 (2010). DOI DOI:10.1016/j.websem.2010.08.001. URL <http://www.sciencedirect.com/science/article/B758F-50THXFH-1/2/84c276a928b5889c9870f9e57eda2658> 768  
769  
770  
771
23. Harth, A.: The billion triple challenge datasets. <http://km.aifb.kit.edu/projects/btc-2011/> (2011) 772
24. Harth, A., Hogan, A., Umbrich, J., Decker, S.: Swse: objects before documents! *Proceedings of the Semantic Web Challenge 2008*, 2008 773  
774
25. Heath, T., Bizer, C.: Linked Data – Evolving the Web into a Global Data Space. Morgan and Claypool Publishers, Seattle, WA USA (2011). URL <http://linkeddatabook.com/> 775  
776
26. Heath, T., Motta, E.: Revyu: linking reviews and ratings into the web of data. *J. Web Semant. Sci. Serv. Agents World Wide Web* **6**(4) (2008) 777  
778
27. Hepp, M.: Goodrelations: an ontology for describing products and services offers on the web. *Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management*, Acirezza, Italy, 2008 779  
780  
781
28. Isele, R., Harth, A., Umbrich, J., Bizer, C.: Ldspider: an open-source crawling framework for the web of linked data. *ISWC 2010 posters and demonstrations track: collected abstracts vol. 658*, 2010 782  
783  
784
29. Jacobs, I., Walsh, N.: Architecture of the World Wide Web, Volume One (2004). <http://www.w3.org/TR/webarch/> 785  
786
30. Jentzsch, A., Hassanzadeh, O., Bizer, C., Andersson, B., Stephens, S.: Enabling tailored therapeutics with linked data. *Proceedings of the WWW2009 Workshop on Linked Data on the Web*, 2009. URL [http://events.linkeddata.org/lidow2009/papers/lidow2009\\\_\\\_paper9.pdf](http://events.linkeddata.org/lidow2009/papers/lidow2009\_\_paper9.pdf) 787  
788  
789

31. Klyne, G., Carroll, J.J.: Resource description framework (RDF): concepts and abstract syntax – W3C recommendation (2004). [Http://www.w3.org/TR/rdf-concepts/](http://www.w3.org/TR/rdf-concepts/) 790  
791
32. Kobilarov, G., Scott, T., Raimond, Y., Oliver, S., Sizemore, C., Smethurst, M., Bizer, C., Lee, R.: Media meets semantic web – how the bbc uses dbpedia and linked data to make connections. The semantic web: research and applications, 6th European semantic web conference, 2009, pp. 723–737 792  
793  
794  
795
33. Madhavan, J., Shawn, J.R., Cohen, S., Dong, X., Ko, D., Yu, C., Halevy, A.: Web-scale data integration: you can only afford to pay as you go. Proceedings of the Conference on Innovative Data Systems Research, 2007 796  
797  
798
34. Mika, P.: Microformats and RDFa deployment across the Web. <http://tripletalk.wordpress.com/2011/01/25/rdfa-deployment-across-the-web/> (2011) 799  
800
35. Möller, K., Heath, T., Handschuh, S., Domingue, J.: Recipes for semantic web dog food – the eswc and iswc metadata projects. Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference, Busan, Korea, 2007 801  
802  
803
36. Myers, J.: Creating local visibility to open box products with front-end semantic web. <http://jay.beweepl.com/2010/03/30/creating-local-visibility-to-open-box-products-with-front-end-semantic-web/> (2010) 804  
805  
806
37. Neubert, J.: Bringing the “thesaurus for economics” on to the web of linked data. Proceedings of the WWW2009 Workshop on Linked Data on the Web, 2009 807  
808
38. Sheridan, J., Tennison, J.: Linking uk government data. Proceedings of the WWW2010 Workshop on Linked Data on the Web, 2010. URL <http://ceur-ws.org/Vol-628/lidow2010\paper14.pdf> 809  
810  
811
39. Søren, A., Jens, L., Sebastian, H.: Linkedgeodata – adding a spatial dimension to the web of data. Proceedings of the International Semantic Web Conference, 2009 812  
813
40. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: Williamson C.L., Zurko M.E., Patel-Schneider P.F., Shenoy P.J. (eds.) Proceedings of the 16th International Conference on World Wide Web, ACM, Banff, Alberta, Canada, 8–12 May 2007, pp. 697–706, URL <http://doi.acm.org/10.1145/1242572.1242667> 814  
815  
816  
817
41. Tummarello, G., Cyganiak, R., Catasta, M., Danielczyk, S., Delbru, R., Decker, S.: Sig.ma: live views on the web of data. Web Semant. Sci. Serv. Agents World Wide Web 8(4), 355–364 (2010). DOI DOI:10.1016/j.websem.2010.08.003. URL <http://www.sciencedirect.com/science/article/B758F-50THXFH-3/2/50a61206ad3a34d5541aebf5a465484a> 818  
819  
820  
821
42. Van de Sompel, H., Lagoze, C., Nelson, M., Warner, S., Sanderson, R., Johnston, P.: Adding escience assets to the data web. Proceedings of the 2nd Workshop on Linked Data on the Web (LDOW2009), 2009 822  
823  
824

**AUTHOR QUERIES**

- AQ1. Kindly check the corresponding author.
- AQ2. Please confirm the changes in the sentence “In parallel to the different...”
- AQ3. Please check if ‘Microformats deployment’ can be changed to ‘Microformat deployment.’ Please update in other occurrences also.
- AQ4. Please confirm the changes in the sentence “Present deployment statistics...”
- AQ5. Please confirm the changes in the sentences “Microformats (also referred...” and “Each Microformat...”
- AQ6. Please check if ‘Microformats community’ can be changed as ‘Microformat community’. Please update in other occurrences also.
- AQ7. Please check the sentence starting ”In 2004 followed...” for completeness.
- AQ8. Please check if ‘Microformats Syntax’ can be changed as ‘Microformat Syntax’. Please update in other occurrences also. Please check and update the same for ‘microformats technology’ also.
- AQ9. Please check if ‘microformats approach’ can be changed as ‘microformat approach’.
- AQ10. Please confirm the changes in the sentence “a collection of vocabularies for...”
- AQ11. In the sentence “...in efforts related to the data...”, please check if ‘which’ can be inserted following ‘in’.
- AQ12. Please check if ‘interlink library catalogs’ can be changed as interlinking library catalogs’.
- AQ13. Please confirm the changes in the sentences “This conference track aims...” and “Detailed Statistics are provided...”
- AQ14. Please confirm the changes in the sentence “...data that is embedded...”
- AQ15. Kindly provide the accessed date for [1, 2, 4, 5, 12, 15–18, 20, 23, 29, 31, 34, 36].

## Chapter 2

# Storing and Indexing Massive RDF Datasets

Yongming Luo, François Picalausa, George H.L. Fletcher, Jan Hidders,  
and Stijn Vansumeren

### 2.1 Introduction: Different Perspectives on RDF Storage

The resource description framework (RDF for short) provides a flexible method for modeling information on the Web [34, 40]. All data items in RDF are uniformly represented as triples of the form  $(subject, predicate, object)$ , sometimes also referred to as  $(subject, property, value)$  triples. As a running example for this chapter, a small fragment of an RDF dataset concerning music and music fans is given in Fig. 2.1.

Spurred by efforts like the Linking Open Data project [35, 75], increasingly large volumes of data are being published in RDF. Notable contributors in this respect include areas as diverse as the government [11], the life sciences [9], Web 2.0 communities, and so on.

To give an idea of the volumes of RDF data concerned, as of September 2010, there are 28,562,478,988 triples in total published by data sources participating in the Linking Open Data project [5]. Many individual data sources (like, e.g., PubMed, DBpedia, MusicBrainz) contain hundreds of millions of triples (797, 672, and 179 millions, respectively). These large volumes of RDF data motivate the need for scalable native RDF data management solutions capable of efficiently storing, indexing, and querying RDF data.

In this chapter, we present a general and up-to-date survey of the current state of the art in RDF storage and indexing.

It is important to note that RDF data management has been studied in a variety of contexts. This variety is actually reflected in a richness of the perspectives and approaches to storage and indexing of RDF datasets, typically driven by particular classes of query patterns and inspired by techniques developed in various research

---

AQ1

Y. Luo (✉) · F. Picalausa · G.H.L. Fletcher · J. Hidders · S. Vansumeren  
Department of Mathematics and Computer Science, Eindhoven University of Technology,  
P.O. Box 513, 5600 MB Eindhoven, The Netherlands  
e-mail: [y.luo@tue.nl](mailto:y.luo@tue.nl); [g.h.l.fletcher@tue.nl](mailto:g.h.l.fletcher@tue.nl)

```

{<work5678, FileType, MP3 >,
<work5678, Composer, Schoenberg >,
<work1234, MediaType, LP >,
<work1234, Composer, Debussy >,
<work1234, Title, La Mer >,
<user8604, likes, work5678 >,
<user8604, likes, work1234 >,
<user3789, name, Umi >,
<user3789, birthdate, 1980 >,
<user3789, likes, work1234 >,
<user8604, name, Teppei >,
<user8604, birthdate, 1975 >,
<user8604, phone, 2223334444 >,
<user8604, phone, 5556667777 >,
<user8604, friendOf, user3789 >,
<Debussy, style, impressionist>,
<Schoenberg, style, expressionist>, ...}

```

**Fig. 2.1** A small fragment of an RDF dataset concerning music fans

communities. In the literature, we can identify three basic perspectives underlying this variety. 28  
29

### 2.1.1 The Relational Perspective

30

The first basic perspective, put forward by the database community, is that an RDF graph is just a particular type of relational data, and that techniques developed for storing, indexing, and answering queries on relational data can hence be reused and specialized for storing and indexing RDF graphs. In particular, techniques developed under this perspective typically aim to support the full SPARQL language. 31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49

The most naive approach in this respect is simply to store all RDF triples in a single table over the relation schema (*subject, predicate, object*). An important issue in this approach is that, due to the large size of the RDF graphs and the potentially large number of self-joins required to answer queries, care must be taken to devise an efficient physical layout with suitable indexes to support query answering. In addition to this simple *vertical representation*, there also exists a *horizontal representation* where the triple predicate values are interpreted as column names in a collection of relation schemas. For example, we can divide the music fan dataset of Fig. 2.1 into five relations: Works, Users, Composers, Likes, and FriendOf; the Users relation would have columns Name, BirthDate, and Phone. Major issues here are dealing with missing values (e.g., subject work5678 does not have a title) and multivalued attributes (e.g., user8604 has two phone numbers). In Sect. 2.2, we give an up-to-date survey of the literature on the relational perspective, thereby complementing earlier surveys focusing on this perspective [36, 50, 62, 64–66, 69].

### 2.1.2 The Entity Perspective

50

The second basic perspective, originating from the information retrieval community, is resource centric. Under this *entity perspective*, resources in the RDF graph are interpreted as “objects,” or “entities.” Similarly to the way in which text documents are determined by a set of keyword terms in the classical information retrieval setting, each entity is determined by a set of attribute–value pairs in the entity perspective [18, 78]. In particular, a resource  $r$  in RDF graph  $G$  is viewed as an entity with the following set of (attribute, value) pairs:

$$\text{entity}(r) = \{(p, o) \mid (r, p, o) \in G\} \cup \{(p^{-1}, o) \mid (o, p, r) \in G\}.$$

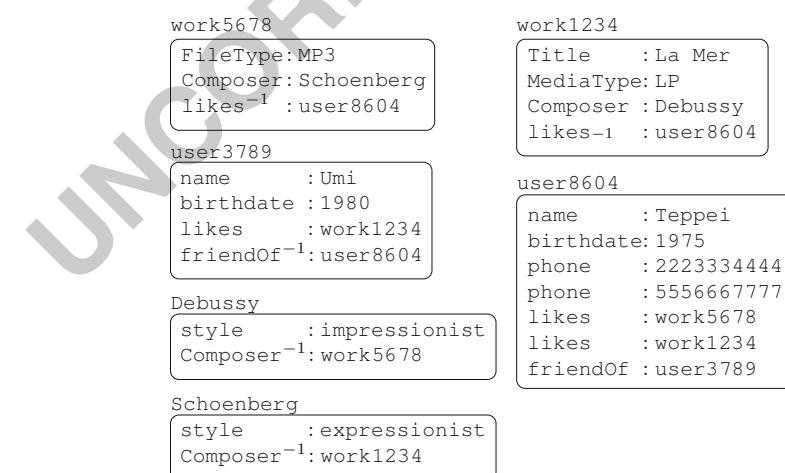
Figure 2.2 illustrates this idea for the RDF graph in Fig. 2.1. Techniques from the information retrieval literature can then be specialized to support query patterns that retrieve entities based on particular attributes and/or values [18]. For example, we have in Fig. 2.2 that user8604 is retrieved when searching for entities born in 1975 (i.e., have 1975 as a value on attribute birthdate) as well as when searching for entities with friends who like Impressionist music. Note that entity user3789 is not retrieved by either of these queries. Further discussion and a survey of the literature on this perspective are given in Sect. 2.3.

65

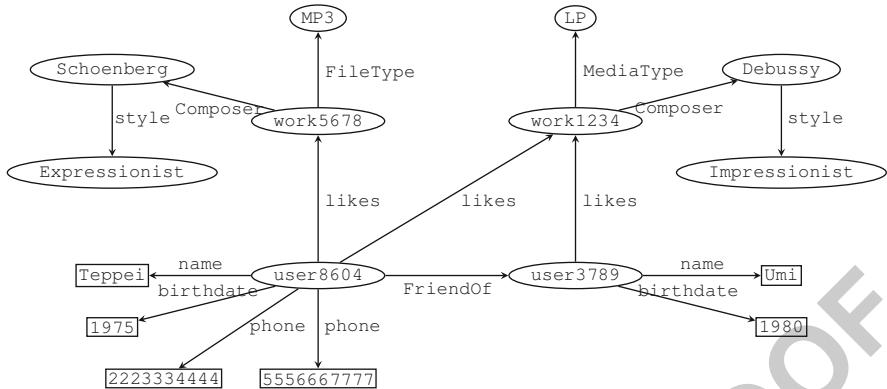
### 2.1.3 The Graph-Based Perspective

66

The third basic perspective, originating from research in semistructured and graph databases, relaxes the entity-centric interpretation of resources in the entity



**Fig. 2.2** The entity view of the music dataset in Fig. 2.1



**Fig. 2.3** The graph view of the music dataset in Fig. 2.1

view, focusing instead on the structure of the RDF data. Under this *graph-based perspective*, the focus is on supporting navigation in the RDF graph when viewed as a classical graph in which subjects and objects form the nodes, and triples specify directed, labeled edges. Under this perspective, the RDF graph of Fig. 2.1 is hence viewed as the classical graph depicted in Fig. 2.3. Typical query patterns supported in this perspective are graph-theoretic queries such as reachability between nodes. For example, in the graph of Fig. 2.3, we might be interested to find those users which have a fan of Debussy somewhere in their social network (i.e., in the friendOf subgraph). The major issue under this perspective is how to explicitly and efficiently store and index the implicit graph structure. Further details and a survey of the literature on the graph-based perspective are given in Sect. 2.4.

### 2.1.4 Proviso

We note that recent research has devoted considerable effort to the study of managing massive RDF graphs in distributed environments such as P2P networks [58] and the MapReduce framework [63]. In this chapter, in contrast, we focus on storage and indexing techniques for a single RDF graph instance in a non-distributed setting. In addition, we focus on the storage and indexing of *extensional* RDF data only, and refer to other surveys [43, 50, 62, 69] and Chap. 10 for an overview of the state of the art in inferring and reasoning with the *intentional* data specified by RDF ontologies.

We assume that the reader is familiar with the basic concepts and terminology of the relational data model, as well as basic concepts of the RDF data model [34, 40], and the SPARQL query language for RDF [59].

There are obviously tight connections between each of the above perspectives since they are interpretations of the same underlying data model. It is therefore sometimes difficult to clearly identify to which perspective a particular storage

2 Storing and Indexing Massive RDF Datasets	33
or indexing approach has the most affinity. Where appropriate, we indicate such “hybrids” in our detailed survey of the literature, to which we turn next.	95 96

## 2.2 Storing and Indexing Under the Relational Perspective 97

Conceptually, we can discern two different approaches for storing RDF data in relational databases. The *vertical representation* approach stores all triples in an RDF graph as a single table over the relation schema (*subject*, *predicate*, *object*). The *horizontal representation* approach, in contrast, interprets triple predicate values as column names, and stores RDF graphs in one or more wide tables. Additional indexes can be built on top of these representations to improve performance. We survey both representations next, and conclude with indexing. 98  
99  
100  
101  
102  
103  
104

### 2.2.1 A Note on the Storage of IRIs and Literal Values 105

Before we dive into the details of the vertical and horizontal representations, it is important to note that both approaches generally make special provisions for storing RDF resources efficiently. Indeed, rather than storing each IRI or literal value directly as a string, implementations usually associate a unique numerical identifier to each resource and store this identifier instead. There are two motivations for this strategy. First, since there is no *a priori* bound on the length of the IRIs or literal values that can occur in RDF graphs, it is necessary to support variable-length records when storing resources directly as strings. By storing the numerical identifiers instead, fixed-length records can be used. Second, and more importantly, RDF graphs typically contain very long IRI strings and literal values that, in addition, are frequently repeated in the same RDF graph. The latter is illustrated in Fig. 2.1, for example, where user8604 is repeated eight times. Since this resource would be represented by an IRI like <http://www.example.org/users/user8604> in a real-world RDF graph, storing the short numerical identifiers hence results in large space savings. 106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120

Unique identifiers can be computed in two general ways: 121

- Hash-based approaches obtain a unique identifier by applying a hash function to the resource string [30, 31, 64], where the hash function used for IRIs may differ from the hash function used for literal values. This is the approach taken by 3-Store [31], for example, where the hash functions are chosen in such a way that it is possible to distinguish between hash values that originate from IRIs and literal values. Of course, care must be taken to deal with possible hash collisions. In the extreme, the system may reject addition of new RDF triples when a collision is detected [31]. 122  
123  
124  
125  
126  
127  
128  
129

To translate hash values back into the corresponding IRI or literal value when answering queries, a distinguished *dictionary* table is constructed. 130  
131

- Counter-based approaches obtain a unique identifier by simply maintaining a counter that is incremented whenever a new resource is added. To answer queries, dictionary tables that map from identifiers to resources and vice versa are constructed [7, 32, 33, 56, 57]. Typically, these dictionary tables are stored as BTrees for efficient retrieval. 132  
133  
134  
135  
136

A variant on this technique that is applicable when the RDF graph is static is to first sort the resource strings in lexicographic order, and to assign the identifier  $n$  to the  $n$ th resource in this order. In such a case, a single dictionary table suffices to perform the mapping from identifiers to resources and vice versa [77]. 137  
138  
139  
140

Various optimizations can be devised to further improve storage space. For example, when literal values are small enough to serve directly as unique identifiers (e.g., literal integer values), there is no need to assign unique identifiers, provided that the storage medium can distinguish between the system-generated identifiers and the small literal values [15, 16]. Also, it is frequent that many IRIs in an RDF graph share the same namespace prefix. By separately encoding this namespace prefix, one can further reduce the storage requirements [15, 16, 49]. 141  
142  
143  
144  
145  
146  
147

## 2.2.2 Vertical Representation

148

In the vertical representation (sometimes also called *triple store*), triples are conceptually stored in a single table over the relation schema (*subject*, *predicate*, *object*) [7, 10, 30, 31, 80]. Some implementations include an additional *context* column in order to store RDF *datasets* rather than single RDF graphs. In this case, the *context* column specifies the IRI of the named graph in which the RDF triple occurs. 149  
150  
151  
152  
153  
154

An important issue in the vertical representation is that, due to the large size of the RDF graphs and the potentially large number of self-joins required to answer queries, care must be taken to devise an efficient physical layout with suitable indexes to support query answering [15, 32, 33, 41, 49, 56, 57, 67, 79]. 155  
156  
157  
158

### 2.2.2.1 Unclustered Indexes

159

One of the first proposals for addressing this scalability issue compared the impact of adding the following four different sets of BTree indexes on the triple table [49]: 160  
161  
162

1. The first set of unclustered indexes consists of an index on the subject column ( $s$ ) alone, an index on the property ( $p$ ) column alone, and index on the object column ( $o$ ) alone. 163  
164  
165
2. The second set of unclustered indexes consists of a combined index on subject and property ( $sp$ ), as well as an index on the object column ( $o$ ) alone. 166  
167

2 Storing and Indexing Massive RDF Datasets	35
3. The third set of unclustered indexes consists of a combined index on property and object ( <i>po</i> ).	168 169
4. The final set has a combined clustered index on all columns together ( <i>spo</i> ).	170

The authors note that, for an ad hoc set of queries and on a specific RDF graph, the first set of indexes drastically improves the SQL-based query processing performance, whereas the other index sets achieve less performance.

More recently, Kolas et al. extend the idea of indexing each of the subject, predicate, and object columns separately [41]. In their Parliament database system, each RDF resource  $r$  (IRI or literal) is mapped, by means of a BTree, to a triple of pointers ( $p_1, p_2, p_3$ ). Here,  $p_1$  points to the first record in the triple table in which  $r$  occurs as a subject,  $p_2$  points to the first record in which  $r$  occurs as a predicate, and similarly  $p_3$  points to the first record in which  $r$  occurs as an object. In addition, each record in the triple table is extended with three additional pointers that point to the next triple in the table containing the same subject, predicate, and object, respectively. In this way, a linked list is constructed. While this layout consumes very little storage space, traversing large linked lists may incur a nonnegligible disk I/O cost.

### 2.2.2.2 Clustered BTree Indexes

Whereas the above indexes are *unclustered* and hence simply store pointers into the triple table (avoiding storing multiple copies of the RDF triples), there are many triple stores that aggressively store the vertical representation table in multiple sorted orders according to various permutations of the sequence (*context, subject, predicate, object*) and its subsequences [10, 15, 56, 57, 79]. Usually, a *clustered* BTree is constructed for each of the selected permutations for efficient retrieval; the desired triple ordering is then available in the leaves of the BTree. While these approaches are more demanding in terms of storage space, they support efficient query answering since the availability of the various sorted versions of the triple table enables fast merge joins.

- Weiss et al. [79] consider this approach in their HexaStore engine by creating six different clustered BTree indexes: *spo*, *sop*, *pso*, *pos*, *osp*, and *ops*. Additionally, common payload spaces are shared between indexes. For example, the *spo* and *pso* indexes share a common materialized set of associated *o* values. In this way, some storage redundancy is eliminated. Due to the remaining redundancy in the indexes, efficient retrieval of both fully and partially specified triples is supported. For example, it suffices to consult the *sop* index to retrieve all triples with subject `User8604` and object `user3789`. Recently, Wang et al. proposed to implement these indexes by means of a column-oriented database instead of using BTrees [77].
- Neumann and Weikum [56, 57] take this approach further in their RDF-3X engine by adding, to the six indexes above, the so-called *projection indexes* for each strict subset of  $\{\text{subject}, \text{predicate}, \text{object}\}$ , again in every order. This adds

additional nine indexes:  $s$ ,  $p$ ,  $o$ ,  $sp$ ,  $ps$ ,  $so$ ,  $os$ ,  $op$ , and  $po$ . Instead of storing 209 triples, the projection indexes conceptually map search keys to the number of 210 triples that satisfy the search key. For example, the projection index on subject 211 alone maps each subject  $s$  to the cardinality of the multiset 212

$$\{ | (p, o) \mid (s, p, o) \text{ occurs in the RDF Graph} | \}.$$

The projection indexes are used to answer aggregate queries, to avoid computing 213 some intermediate join results, and to provide statistics that can be used by the 214 RDF-3X cost-based query optimizer. RDF-3X also includes advanced algorithms 215 for estimating join cardinalities to facilitate query processing. 216

To reduce the storage space required, RDF-3X in addition uses a compression 217 scheme at the BTree leaf block level. The idea behind this compression scheme 218 is simple: in a given index sort order, say  $spo$ , it is likely that subsequent triples 219 share a common subject or subject–predicate prefix. In this case, these repetitions 220 are not stored. 221

- The widely used Virtuoso DBMS and RDF engine also stores different permutations of  $spoc$  [15, 16] (where  $c$  indicates the context column). By default, 222 however, only clustered BTree indexes on  $cspo$  and  $ocps$  are materialized, 223 although other indexes can be enabled by the database administrator. The storage 224 cost for these indexes is reduced by introducing compression schemes at the data 225 level, as well as at the page level. 226

In addition, Erling and Mikhailov [16] report that in practical RDF graphs, 228 many triples share the same predicate and object. To take advantage of this 229 property, Virtuoso builds bitmap indexes for each  $opc$  prefix by default, storing 230 the various subjects. McGlothlin et al. build on this idea and use bitmap indexes 231 instead of BTrees as the main storage structure [53]. 232

Furthermore, Atre et al. [4] build a virtual 3D bitmap upon an RDF graph, with 233 each dimension representing the subjects, predicates, and objects, respectively. 234 Each cell of this 3D bitmap hence states the existence of one combination of 235 the coordinates, i.e., a triple. By slicing the bitmap from different dimensions, 236 several 2D bit matrices (BitMats) are generated, essentially indicating various 237 permutations of the triples. These so-called BitMats are compressed and stored 238 on disk, along with a metafile maintaining summary and location information. Bit 239 operations on this metafile are used to filter out BitMats during join evaluation, 240 thereby accelerating query evaluation. 241

- Finally, Fletcher and Beck [17] propose a hybrid of the unclustered and clustered 242 approaches, wherein a single BTree index is built over the set of all resources 243 appearing in the dataset. The payload for a resource  $r$  consists of three sorted 244 lists, namely  $op$  where  $r$  appears as a subject,  $so$  where  $r$  appears as a predicate, 245 and  $sp$  where  $r$  appears as an object. The advantage of this so-called TripleT 246 approach is simplicity of design and maintenance, as well as increased locality 247 of data, in the sense that all information pertaining to a resource is available in a 248 single local data structure. The TripleT index was shown to be competitive with 249 the previous approaches, in terms of both storage and query evaluation costs. 250

### 2.2.2.3 Variations on the Clustered Index Approach

251

Predating the above approaches, Harth et al. [32, 33] considered clustered BTrees indexing of *sposc* quads in their YARS system. Since they were not interested in answering queries involving joins, only 6 of the 16 subsets of *sposc* are materialized: *sposc*, *poc*, *ocs*, *csp*, *cp*, *os*. In a more recent work, Harth et al. alternatively consider the use of extensible hash tables and in-memory sparse indexes instead of BTrees [33]. They observe that while disk I/O decreases from  $O(\log(n))$  to  $O(1)$  to answer queries on  $n$  quads by using extensible hash tables rather than BTrees, 16 hash tables are required to mimic the behavior of the six BTrees. As an alternative to hash tables, Harth et al. note that a similar decrease in disk I/O can be obtained by constructing an in-memory sparse index for each of the six subsets of *sposc*. Each entry in a sparse index points to the first record of a sorted block on disk. Evidently, there is a trade-off between main memory occupation and performance: to accommodate larger RDF graphs or to reduce main-memory consumption, the index can be made sparser by making the sorted blocks larger, at the expense of more disk I/O.

Wood et al. [82] build on this idea by representing the six sparse indexes on *sposc* using AVL trees [82]. For performance reasons, nodes of the AVL tree point to separate pages of 256 ordered triples. Each node of the AVL tree stores the first and the last triples of the page it represents to enable traversal of the tree without accessing the pages.

### 2.2.3 Horizontal Representation

272

Under the horizontal representation, RDF data are conceptually stored in a single table of the following format: the table has one column for each predicate value that occurs in the RDF graph and one row for each subject value. For each  $(s, p, o)$  triple, the object  $o$  is placed in the  $p$  column of row  $s$ . The music fan data from Fig. 2.1 would hence be represented by the following table.

subject	FileType	Composer	...	phone	friendOf	style
work5678	MP3	Schoenberg				
work1234		Debussy				
...						
user8604				{2223334444, user3789 5556667777}		
Debussy					impressionist	
Schoenberg					expressionist	

278

As can be seen from this example, it is rare that a subject occurs with all possible predicate values, leading to sparse tables with many empty cells. Care must hence be taken in the physical layout of the table to avoid storing the empty cells. Also, since it is possible that a subject has multiple objects for the same predicate (e.g., user8604 has multiple phone numbers), each cell of the table represents in principle a *set* of objects, which again must be taken into account in the physical layout.

### 2.2.3.1 Property Tables

To minimize the storage overhead caused by empty cells, the so-called *property-table approach* concentrates on dividing the wide table in multiple smaller tables containing related predicates [46, 68, 80]. For example, in the music fan RDF graph, a different table could be introduced for `Works`, `Fans`, and `Artists`. In this scenario, the `Works` table would have columns for `Composer`, `FileType`, `MediaType`, and `Title`, but would not contain the unrelated `phone` or `friendOf` columns. The following strategies have been proposed to identify related predicates:

- In the popular Jena RDF, the grouping of predicates is defined by applications [80]. In particular, the application programmer must specify which predicates are multivalued. For each such multivalued predicate  $p$ , a new table with schema (`subject`,  $p$ ) is created, thereby avoiding replicating other values. Jena also supports so-called *property-class* tables, where a new table is created for each value of the `rdf:type` predicate. Again, the actual layout of each property-class table is application defined. The remaining predicates that are not in any defined group are stored independently.
- In RDFBroker, Sintek et al. [68] identify related predicates by computing, for every subject  $x$  in the RDF graph  $G$ , the set of predicates  $P_x = \{p \mid \exists o, ((x, p, o) \in G)\}$ . For each such set  $P_x$ , a corresponding predicate table over the relational schema  $\{\text{subject}\} \cup P_x$  is created. Queries can be answered by first identifying the property tables that have all of the predicate values specified in the query, answering the query on these tables, and then taking the union of the results.

Sintek et al. note that this approach actually creates many small tables, which is harmful for query evaluation performance. To counter this, they propose various criteria for merging small tables into larger ones, introducing NULL values when a predicate is absent.

- Levandoski et al. [46] leverage previous work on association rule mining to automatically determine the predicates that often occur together, through a methodology they call data-centric storage. The methodology aims at maximizing the size of each group of predicates, while minimizing the number of NULL values that will occur in the tables. The remaining predicates that are not in any group are stored independently.

**2.2.3.2 Vertical Partitioning**

320

The so-called *vertically partitioned database approach* (not to be confused with the vertical representation approach of Sect. 2.2.2) takes the decomposition of the horizontal representation to its extreme [1]: each predicate column  $p$  of the horizontal table is materialized as a binary table over the schema (*subject*,  $p$ ). Each row of each binary table essentially corresponds to a triple. Note that, hence, both the empty cell issue and the multiple object issue are solved at the same time. Abadi et al. [1] and Sidiropoulos [67] note that the performance of this approach is best when sorting the binary tables lexicographically according to (*subject*,  $p$ ) to allow fast joins; and that this approach is naturally implemented by a column-oriented database.

321  
322  
323  
324  
325  
326  
327  
328  
329  
330**2.2.3.3 Disadvantages and Advantages**

331

It has been observed that, in both approaches, the distinction between predicate values (which are elevated to the status of column names) and subject and object values (which are stored as normal values) is a weakness when answering queries that do not specify the predicate value. For such queries, the whole horizontal table (or all of its partitions) must be analyzed. This hinders performance, especially in the presence of many predicates values. Moreover, the relational schema must be changed whenever a new predicate value is added to the RDF graph. This is not well supported in relational database management systems, where relation schemas are traditionally considered to be static.

332  
333  
334  
335  
336  
337  
338  
339  
340

On the positive side, the horizontal representation makes it easy to support typing of object values (e.g., it becomes possible to declare that object values of predicate *age* are integers, whereas object values of predicate *birthdate* are dates). Moreover, it is easy to integrate existing relational data with RDF data: it suffices to consider the key of a relational table as the subject and the remaining columns as predicates. As such, the existing table essentially becomes a new RDF graph [80].

341  
342  
343  
344  
345  
346  
347**2.2.4 More on Indexing**

348

While some of the storage strategies presented in Sects. 2.2.2 and 2.2.3 already provide storage strategy-dependent indexes, additional indexes may be desired to speed up query processing even further.

349  
350  
351**2.2.4.1 Literal-Specific Indexes**

352

A first line of work in this respect targets the indexation of the *literals* occurring in an RDF graph. To support full-text search, for example, one can add inverted indexes

353  
354

on string literals [15, 32], or N-grams indexes [45] to support regular expression search. Similarly, if it is known that particular literals hold geographical information (e.g., latitude and longitude), specialized spatial indexes such as kd- and quad-trees can be added [42].

#### 2.2.4.2 Join Indexes

359

A second line of work extends the classical idea of join indexes [74] to the RDF setting. In particular, Chong et al. [10] propose the construction of six binary join indexes to speed up self-joins in the vertical representation storage approach: there is an index for the self-join on subject–subject, on subject–predicate, on subject–object, on predicate–predicate, on predicate–object, and on object–object. In this way, a SPARQL query that expresses a join between two triple patterns, like

```
SELECT ?work
WHERE {
    ?user <likes> ?work .
    ?work <Composer> "Debussy" .
}
```

can be answered using the object–subject join index.

AQ2

Groppe et al. [25] extend this idea further. They not only remark in particular that triple patterns in SPARQL queries rarely consist of variables only but also mention IRIs or literals (as in the example above). In order to find the joining triples that satisfy the mentioned IRI/literal requirements, Groppe et al. extend the notion of a join index and store, for each of the possible self-joins listed above (subject–subject, subject–predicate,...): 16 different indexes. Conceptually, each of the latter indexes map sequences of IRIs and literals that do not occur in the join position to the pair of joining triples.

To illustrate, consider the join index on subject–object which is meant to aid in answering joins of two triple patterns, say  $(?x, p_1, o_1)$  and  $(s_2, p_2, ?x)$  like in the example query above. Groppe et al. construct the 16 indexes that map all subsets of  $p_1, o_1, s_2, p_2$  to the joining triples. So, in an RDF graph consisting of the triples

<i>tid</i>	subject	predicate	object
$t_1$	$a$	$b$	$c$
$t_2$	$d$	$e$	$a$
$t_3$	$f$	$g$	$d$

the 16 indexes would be constructed as follows:

$p_1 \ o_1 \ s_2 \ p_2$	$p_1 \ s_2 \ p_2$	$p_1$
$b \ c \ d \ b$	$(t_1, t_2)$	$b \   (t_1, t_2)$
$b \ a \ f \ g$	$(t_2, t_3)$	$b \   (t_2, t_3)$

This approach can be further extended to handle joins between more than two triple patterns. 385  
386

#### 2.2.4.3 Materialized Views

387

Finally, a third line of work applies the classical notion of answering queries using views [28] to the RDF setting, thereby also extending the work on join indexes discussed above. 388  
389  
390

In a nutshell, when answering queries using views, we are given a query  $Q$  over an RDF graph  $G$ , as well as a set of materialized (relational) views  $V_1, \dots, V_n$  over  $G$ , and we want to find the cheapest query execution plan for  $Q$  that can use both  $G$  and the views  $V_1, \dots, V_n$ . Of course, a critical difficulty in this respect is the selection of the views  $V_1, \dots, V_n$  to materialize. 391  
392  
393  
394  
395

In the RDF\_MATCH approach, Chong et al. [10] propose to materialize so-called *subject–property tables* in addition to the (vertically represented) RDF graph. A subject–property table is similar to the property tables from the horizontal representation storage approach: for a set of subject values and a group of related single-valued predicates, a table is constructed where each predicate occurs as a column name. Each row contains one subject value, along with the object value of each predicate. Contrary to property tables of the horizontal store approach, however, the proposed view here is not the primary way of storage. 396  
397  
398  
399  
400  
401  
402  
403

In the RDFMatView system, Castillo et al. [8] offer the users the possibility to identify and materialize SPARQL basic graph pattern. They note that basic graph patterns can be viewed as conjunctive queries, for which query containment is known to be decidable. When processing a new query, materialized views that can be used to answer the query are selected based on a query containment test. Subsequently, the selected views are assembled to help in the processing. A corresponding cost model to compare alternate rewritings is also proposed. A similar approach uses expression trees of the input expression as the basis for caching intermediate query results [85]. 404  
405  
406  
407  
408  
409  
410  
411  
412

The authors of RDFViewS [22] take the complementary approach of identifying important conjunctive queries in a given weighted workload. An RDF Schema is used to guide the process of selecting the appropriate conjunctive queries. 413  
414  
415

### 2.3 Storing and Indexing Under the Entity Perspective

416

As mentioned earlier in Sect. 2.1, a major alternative way to view an RDF graph is to treat it as a collection of entity descriptions [18, 78]. Similarly to the way in which text documents are viewed as sets of keyword terms in the classical information retrieval setting, each entity is determined by a set of attribute–value and relationship–entity pairs in the entity perspective. Approaches under the entity perspective make heavy use of the inverted index data structure [81]. The reason 417  
418  
419  
420  
421  
422

for this is twofold. First and foremost, inverted indexes have proven to scale to very 423  
large real-world systems. Moreover, many of the aspects of inverted indexes have 424  
been thoroughly studied (e.g., encoding and compression techniques), from which 425  
the new approaches for RDF storage and indexing can benefit. 426

A large part of the work under the entity perspective investigates how to map 427  
the RDF model to traditional information retrieval (IR) systems. Typically, the 428  
following two general types of queries are to be supported: 429

- *Simple keyword queries.* A keyword query returns all entities that contain an 430  
attribute, relationship, and/or value relevant to a given keyword. To illustrate, in 431  
Fig. 2.2, if the keyword query is “work5678,” query evaluation returns the set 432  
of all entities having work5678 somewhere in their description, namely, the 433  
entities work5678, Debussy, and user8604. 434
- *Conditional entity-centric queries.* A conditional entity-centric query returns all 435  
known entities that satisfy some given conditions on a combination of attribute, 436  
relationships, and values at the same time. For Fig. 2.2, the query “retrieve all 437  
entities with the value work5678 on the Composer<sup>-1</sup> attribute” is such a 438  
conditional entity-centric query. This query returns the set of entities having value 439  
work5678 on the Composer<sup>-1</sup> attribute, namely, the entity Debussy. 440

Note that, in contrast to relational approaches, query results under the entity 441  
perspective return a subset of the known entities, rather than relational tables. In this 442  
respect, the relational queries are more powerful since they allow the restructuring 443  
and combination of data (e.g., by performing joins, etc.), whereas the entity-centric 444  
queries only return data in their existing form. It should be noted, however, that set 445  
operations (intersection, union, etc.) are usually supported in the entity perspective 446  
to further manipulate the returned set of entities[18]. A nice comparison of index 447  
maintenance costs between relational and entity perspective systems is presented 448  
in [12]. 449

In the rest of this section, we will introduce some of the representative works in 450  
the literature. Note that this is not an exhaustive survey of this rapidly advancing 451  
research area, but rather a selection of recent systems that illustrate the basic 452  
concepts of storage and indexing RDF data under the entity perspective. 453

### 2.3.1 Dataspaces

454

Dong and Halevy [13] were the first to consider inverted indexes for answering 455  
entity-centric queries over RDF graphs, in the context of the so-called *dataspace* 456  
*support system paradigm*. First proposed by Franklin et al. [20, 29], the dataspaces 457  
paradigm constitute a gradual, pay-as-you-go, approach to the integration of 458  
information from across diverse, interrelated, but heterogeneous data sources. In 459  
this setting, dataspaces should also provide query support in a pay-as-you-go setting, 460  
depending on the querying capabilities of the data sources themselves. For example, 461  
if a data source is plain text, maybe only keyword search is enabled. If it is 462

semistructured data, a corresponding structured query language (comparable to XQUERY or SPARQL) could be supported. This also requires the underlying index structure to be flexible enough to accommodate the query language.

While Dong and Halevy [13] do not explicitly use the terms *RDF data model* and *RDF graphs*, the data model that they use is equivalent to RDF. Specifically, in their paper, data from different sources are unified in triples of the form  $(entity, attribute, value)$  or  $(entity, association, entity)$ . These triples are then passed to the indexer for further processing. The proposed system supports both simple keyword queries and conditional entity-centric queries, and always returns a collection of entities. The system also provides support for synonyms and hierarchies, which support the discovery of more relevant results than plain querying. Referring to Fig. 2.2, for example, if the (static) system-defined hierarchy indicates that `MediaType` is a superclass of `FileType`, then the evaluation of the conditional entity-centric query “retrieves all entities with value MP3 on the `MediaType` attribute” with hierarchy expansion containing the entity `work1234`.

The basic indexing structure of Dong and Halevy is an inverted index. Each *term* in this index points to an inverted list that contains all the identifiers of all entities that *match* the term. There are three kinds of terms:

- To start with, every value  $v$  is considered a term. An entity  $e$  matches the term  $v$  if there exists a triple  $(entity\ e, attribute\ a, value\ v)$ , for some attribute  $a$ . This is the same as the traditional IR approaches, to provide answering keyword query capability.
- To support conditional entity-centric queries, concatenations of a value  $v$  and attribute  $a$  (denoted  $v//a//$ ) are also considered to be terms. An entity matches such a term if the triple  $(entity\ e, attribute\ a, value\ v)$  is present.
- Moreover, concatenations of value  $v$  and association  $a_1$  ( $v//a_1//$ ) are also considered terms. An entity  $e_1$  matches this term if there exists a pair of triples  $(entity\ e_1, association\ a_1, entity\ e_2)$  and  $(entity\ e_2, attribute\ a_2, value\ v)$  for some entity id  $e_2$  and some attribute  $a_2$ . These kinds of terms are designed to boost a specialized conditional entity-centric query, called the *one-step keyword query*. Such queries are meant to return all entities in which a given keyword appears, as well as entities that are related to entities in which the keyword appears.

For example, when indexing the example dataset in Fig. 2.2, the values (MP3), (Schoenberg), and (user8604) will be treated as terms (among others). The concatenations of values and attributes (MP3//FileType//) and (Schoenberg//Composer//) will also be considered as terms (among others). Then entity `work5678` matches both of these terms, and will hence appear in the inverted list of both terms. Furthermore, because of the triple (user8604, likes, work5678), two associations (user8604, likes, work5678) and (work5678, likes<sup>-1</sup>, user8604) will be generated. For the first association, we will treat (MP3//likes//) and (Schoenberg//likes//) as terms pointing to `user8604`. For the latter association, we let the terms (Teppei//likes<sup>-1</sup>), (1975//likes<sup>-1</sup>), (2223334444//likes<sup>-1</sup>),

and (5556667777//likes<sup>-1</sup>) have the entity work5678 in their inverted lists. 507  
508

Hierarchies are supported using three methods: duplication, hierarchy path, and 509  
a hybrid of these two. The idea is to materialize some of the queries as terms 510  
beforehand, so that query answering will be accelerated. The system has a built- 511  
in synonym table. By using this table, synonymous attributes are transformed to a 512  
canonical one in the index, and thereby, related results are naturally returned. 513

Updates are quite expensive on this index, because one entity is distributed over 514  
several inverted lists. The authors suggest to group updates together and do batch 515  
update operations to ease I/O cost. An open issue from this approach is that the 516  
result highly relies on how well the information is extracted and matched with each 517  
other from the data sources, and how well the hierarchy and synonym information 518  
are maintained. These are also open problems in information retrieval research in 519  
general. 520

### 2.3.2 SIREn

521

SIREn is part of the Sindice search engine project from DERI, where the objective 522  
is to index the entire “Web of Data” as a dataspace [12]. In SIREn, data are unified 523  
to a set of quadruples of the form of (*dataset, entity, attribute, value*), upon which a 524  
tree model is built. In this tree, the dataset is the root, then the entity, attribute, and 525  
value are the children of the previous element, respectively. Each node in the tree is 526  
encoded with some encoding scheme (Dewey Order for instance [24]), so that the 527  
relationship between two nodes in a tree can be determined by only looking at the 528  
encoding value. SIREn has a clear definition of logical operators on this tree model. 529  
All query answering is then done by translating these logical operators into physical 530  
operations on a representation of this tree. 531

SIREn extends traditional inverted indexes for physically representing these 532  
trees. This is done mainly by extending the inverted list component of the index. 533  
For indexing in SIREn, the search term could be the identifier of the dataset, the 534  
entity, the attribute, or the value. The corresponding inverted list consists of five 535  
different streams of integers: a list of entity identifiers, of term frequencies, of 536  
attribute identifiers, of value identifiers, and of term positions. Further refinement 537  
is done for each term to reduce the index size. Notice that each step of query 538  
answering does not return a collection of entities, but rather a collection of nodes 539  
in the tree model. Intermediate results can then be used as an input for subsequent 540  
operations. 541

To answer a keyword query, the system performs a lookup of the search term, 542  
then returns related terms with their inverted lists. The keyword query could come 543  
with a specified type, and then only the given type of inverted lists is returned. Partial 544  
matching is supported by the keyword search, such as prefix matching and regular 545  
expression, depending on the organization of the terms. To answer conditional 546  
entity-centric queries, the system first performs a keyword search with a specific 547

type, then a join of two inverted lists is performed. Results can be further refined by 548  
other logical operations, such as projections. 549

Compression techniques are also incorporated in SIREn. Five inverted files 550  
are created complying with the five types of inverted lists discussed above. Each 551  
inverted file is constructed in a block-based fashion, and then various compression 552  
techniques are studied on these files. One limitation of SIREn is that join operations 553  
between data from different data sources are not efficiently supported. 554

### 2.3.3 *Semplore*

555

Semplore [76] is another representative approach of the entity perspective approach, 556  
also using inverted indexes as the storage back end. Semplore works on RDF 557  
datasets, and supports hybrid query combining of a subset of SPARQL and full-text 558  
search. 559

Semplore has three indexes: an ontology index, an entity index, and a textual 560  
index. The ontology index stores the ontology graph of the dataset, including 561  
super-/subconcepts and super-/subrelations. The entity index stores the relationships 562  
between the entities. The textual index handles all triples with the “text” predicate, 563  
enabling the system to do keyword search. The highlight of Semplore is to type 564  
RDF resources as Document, Field, and/or Term concepts in the inverted indexes. 565  
Moreover, the system uses a position list in inverted lists as another dimension of 566  
the inverted index. In particular, for each triple  $(s, p, o)$ , 567

- For the ontology index,  $(p, o)$  is treated as a term and  $s$  is treated as a document. 568  
Here  $p \in \{subConOf, superConOf, subRelOf, superRelOf, type\}$ . 569
- For the entity index, two permutations  $(p, s, o)$  and  $(p^{-1}, o, s)$  are stored. Here 570  
 $p/p^{-1}$  is treated as a term and  $s/o$  is treated as a document.  $o/p$  is stored in 571  
the position list. For example in Fig. 2.2, for entity work5678, the following 572  
mapping is done: 573

Term	Document	Position list
FileType	work5678	MP3
FileType <sup>-1</sup>	MP3	work5678
Composer	work5678	Schoenberg
Composer <sup>-1</sup>	Schoenberg	work5678
likes	user8604	work5678
likes <sup>-1</sup>	work5678	user8604

- For the textual index,  $(p, k)$  is treated as a term, where  $p$  is “text,” and  $k$  is a token 575  
appearing in  $o$ .  $s$  is treated as the document. Following the previous example, if 576  
one more triple (work5678, text, "wonderful description") is 577  
added for entity work5678, then the words wonderful and description 578  
will both be treated as terms, having work5678 in their inverted lists. 579

Semplore has three basic operators: basic retrieval, merge-sort, and mass union. 580  
An input query is translated into these operations and performed on the index. 581  
The main idea is to traverse the query parse tree in a depth first way, combining 582  
intermediate results. 583

To be able to handle incremental updates, Semplore proposes a block-based index 584  
structure, splitting inverted lists into blocks. The Landmark technique [48] is used to 585  
locate a document in one inverted list. When blocks are full, they can split as nodes 586  
split in a BTree. Single update and batch update are both supported in this case. 587

Semplore supports a richer query expressiveness than the dataspace and SIREn 588  
approaches. It can answer queries concerning entity relations. However, this also 589  
involves more I/O cost, where the join operation becomes the dominant operation 590  
in the system. Also due to the incompleteness of the permutations, it is not possible 591  
to answer queries when predicates are missing (e.g., queries like (a,?,b)). 592

### 2.3.4 More on Information Retrieval Techniques

593

Information retrieval techniques are also widely used in other RDF systems. Here 594  
we mention some of them. 595

The most common use of IR techniques is using an inverted index to provide full- 596  
text search functionality. In YARS2 [33], similar to the dataspace approach, for each 597  
triple  $(s, p, o)$ , the index uses the literals appearing in object position ( $o$ ) as terms, 598  
pointing to a list of subjects as the inverted list. Some systems also benefit from 599  
wildcard/prefix search functionality from inverted indexes, for example, [55, 82]. 600  
Natural language processing techniques may also be used for identifying keywords 601  
in the literal strings or URIs. 602

Another common use is object consolidation [72, 78], which is the merging of 603  
resources from different datasets as the same entity. Several techniques such as word 604  
cleaning and synonym merging are applied in this case. 605

Finally, we note that several systems (e.g., [14, 44, 71]) take a hybrid approach 606  
of supporting an entity perspective on the underlying RDF dataset, while translating 607  
search queries into structured queries on the data, typically organized under the 608  
graph-based perspective, which we discuss in the next section. 609

## 2.4 Storing and Indexing Under the Graph-Based Perspective

610

611

As the name *RDF graph* already hints at, the RDF data model can be seen as 612  
essentially a graph-based data model, albeit with special features such as nodes 613  
that can act as edge labels and no fundamental distinction between schemas and 614  
instances, which can be represented in one and the same graph. This graph-based 615  
nature implies that both the types of queries and updates that need to be supported as 616

well as the types of data structures that can be used to optimize these will be similar 617  
to those for graph databases [2]. Typical queries would, for example, be pattern 618  
matching, for example, find an embedding of a certain graph, and path expressions, 619  
such as check if there is a certain type of path between two nodes. Graph databases, 620  
in turn, are similarly closely related to object-oriented and semistructured databases. 621  
Indeed, many ideas and techniques developed earlier for those databases have 622  
already been adapted to the RDF setting. For example, Bönström et al. show in 623  
[6] that RDF can be straightforwardly and effectively stored in an object-oriented 624  
database by mapping both triples and resources to objects. 625

One of the key ideas that has been at the center of much research for graph- 626  
based and semistructured data is the notion of *structural index*. This section is 627  
therefore organized as follows. We first discuss the application of general graph- 628  
based indexing techniques to RDF databases. This is followed by a section on 629  
structural indexes where we first present their historical development for graph- 630  
based and semistructured databases, and then their application to RDF databases. 631

## 2.4.1 General Graph-Based Indexing Methods

632

### 2.4.1.1 Suffix Arrays

633

The problem of matching simple path expressions in a labeled graph can be seen 634  
as a generalization of locating the occurrences of a string as a substring in a larger 635  
target string. A well-known indexing structure for the latter setting is that of the 636  
*suffix array*, which can be described as follows. Assuming that the lexicographically 637  
sorted list of all suffixes of the target string is  $[s_1, \dots, s_n]$ , then a suffix array is an 638  
array  $a[1..n]$  such that  $a[i] = (s_i, j_i)$  with  $j_i$  the starting position of  $s_i$  in the target 639  
string. Given a search string  $s$ , we can quickly find in the array all entries  $1 \leq i \leq n$  640  
where  $s$  is a prefix of  $s_i$ , and therefore,  $j_i$  is a position where  $s$  occurs. This idea 641  
can be generalized to node and edge-labeled directed acyclic graphs as follows. The 642  
suffixes are here defined as alternating lists of node and edge labels that occur on 643  
paths that end in a leaf, i.e., a node with no outgoing edges. With each suffix, the 644  
suffix array then associates, instead of  $j_i$ , a set of nodes in the graph. This approach 645  
is adapted and applied to RDF by Matono et al. in [51] where classes are interpreted 646  
as node labels and predicates as edge labels. The authors justify their approach by 647  
claiming that in practical RDF graphs, cycles are rare, but propose nonetheless an 648  
extension that can deal with cycles by marking in suffixes the labels that represent 649  
repeating nodes. 650

### 2.4.1.2 Tree Labeling Schemes

651

A common technique for optimizing path queries in trees is to label the nodes in 652  
the trees in such a way that the topological relationship between the nodes can be 653

determined by an easy comparison of the labels only. For example, for XML, a common labeling scheme was introduced by Li and Moon [47]. They label each node with a tuple  $(pre, post, level)$ , where  $pre$  is an integer denoting the position of the node in a preorder tree walk,  $post$  is an integer denoting the position of the node in a postorder tree walk, and  $level$  is the level of the node in the tree where the root is the lowest level. Two nodes with labels  $(pr_1, po_1, l_1)$  and  $(pr_2, po_2, l_2)$  will have an ancestor–descendant relationship iff  $pr_1 < pr_2$  and  $po_1 < po_2$ . They have a parent–child relationship iff in addition  $l_2 = l_1 + 1$ . We refer to Gou and Chirkova [24] for an overview of such labeling schemes for XML. In principle, this technique can be also applied to directed acyclic graphs if we transform them into a tree by (1) adding a single new root node above the old root nodes and (2) duplicating nodes that have multiple incoming edges. Clearly this can lead to a very redundant representation, but it can still speed up certain path queries. This approach is investigated by Matono et al. in [52] where, like in [51], it is assumed that RDF graphs are mostly acyclic. Note that for some parts of the RDF graph such as the Class hierarchy and the Predicate hierarchy described by an RDF Schema ontology, this assumption is always correct. As such, these hierarchies are stored separately by Matono et al. [51, 52]. Further study of interval-based labeling schemes for RDF graphs with and without cycles has been undertaken by Furche et al. [21].

#### 2.4.1.3 Distance-Based Indexing

673

In [73], Udrea et al. propose a tree-shaped indexing structure called GRIN index where each node in the tree identifies a subgraph of the indexed RDF graph. This is done by associating with each node a pair  $(r, d)$  with a resource  $r$  and a distance  $d$ , defined as minimal path length and denoted here as  $\delta_G(r, s)$  for the distance between  $r$  and  $s$  in graph  $G$ , which then refers to the subgraph that is spanned by the resources that are within distance  $d$  of  $r$ . The index tree is binary and built such that (1) the root is associated with all resources in the indexed graph, (2) the sets of indexed graph nodes associated with sibling nodes in the tree do not overlap, and (3) the set of nodes associated with a parent is the union of those of its two children. As is shown in [73], such indexes can be built using fairly efficient clustering algorithms similar to partitioning around medoids.

Given a query like a basic graph pattern, we can try to identify the lowest candidate nodes in the tree that refer to subgraphs into which the pattern might match. There are two rules used for this, where we check for query  $q$  and an index node with pair  $(r, d)$ :

1. If  $q$  mentions resource  $s$  and  $\delta_G(r, s) > d$ , then the index node is not a candidate, and neither are all of its descendants.
2. If  $q$  contains also a variable  $v$  and  $\delta_G(r, s) + \delta_q(s, v) > d$ , then the index node is not a candidate and neither are any of its descendants. Note that this rule eliminates viable candidates because it is not necessarily true that if  $v$  is mapped to  $r_v$  by a matching of the query then  $\delta_G(r, s) + \delta_q(s, v) = \delta_G(r, r_v)$ . However, this rule was found to work well in practice in [73].

The query is then executed with standard main-memory-based pattern matching 696  
algorithms on the subgraphs associated with the lowest remaining candidates, i.e., 697  
those that have no parent in the index tree which is a candidate. The restriction to the 698  
lowest is correct if we assume that the basic graph pattern is connected and mentions 699  
at least one resource. 700

#### 2.4.1.4 System $\Pi$

701

Several graph-based indexing techniques were investigated by Wu et al. in the 702  
context of System  $\Pi$ , a hypergraph-based RDF store. In [83], which preceded 703  
system  $\Pi$ , Wu et al. propose to use an index based on Prüfer sequences. These 704  
sequences had already been suggested for indexing XML by Rao and Moon in 705  
[61] and can encode node-labeled trees into sequences such that the problem of 706  
finding embeddings of tree patterns is reduced to finding subsequences. However, 707  
since RDF graphs may contain cycles and have labeled edges, this method needs 708  
to be adapted considerably and loses much of its elegance and efficiency, which is 709  
perhaps why it was not used in System  $\Pi$  as presented by Wu and Li in [83]. A 710  
technique mentioned in both papers is the encoding of the partial orderings defined 711  
by the Class and Property hierarchies in RDF Schema ontologies using a labeling 712  
technique based on prime numbers. First, each node  $n$  in the partial order is assigned 713  
a unique prime number  $p(n)$ . Then, we define  $c(n) = \prod_{m \leq n} p(m)$  where  $\leq$  denotes 714  
the partial order. It then holds that  $n \leq n'$  iff  $c(n')$  is a multiple of  $c(n)$ . 715

#### 2.4.2 Structural Indexes

716

An often recurring and important notion for indexing semistructured data is the 717  
AQ3 *structural index* (or *structure index* or *structural summary*, as it is also known). The 718  
simplest way to explain them is if we assume that the data instance is represented 719  
by a directed edge-labeled graph. An example of a structural index is then a reduced 720  
version of this graph where certain nodes have been merged while maintaining all 721  
edges. In the resulting graph, we store with each node  $m$  the set  $M(m)$  of nodes, 722  
also called the *payload* of  $m$ , which were merged into it. The crucial property of 723  
such an index is that for many graph queries, such as path expressions and graph 724  
patterns, it holds that they can be executed over the structural index and then give 725  
us a set of candidate results that contain the correct results as a subset. For example, 726  
consider the query that returns all pairs of nodes  $(n_1, n_2)$  such that there is a directed 727  
path from  $n_1$  to  $n_2$  that satisfies a certain regular expression. If, when executed over 728  
a structural index, the pair of merged nodes  $(m_1, m_2)$  is returned, then all pairs in 729  
 $M(m_1) \times M(m_2)$  can be considered as candidate results. In fact, for certain types 730  
of queries and merging criteria, it can be shown that the set of candidate results 731  
is always *exactly* the correct result, in which case the index is said to be *precise*. 732  
However, even if this is not the case, in which case the index is called *approximate*, 733

the structural index can provide a quick way of obtaining a relatively small set of candidate solutions, especially if the original graph has to be stored on disk but the index fits in main memory. 734  
735  
736

#### 2.4.2.1 Dataguides, 1-Indexes, 2-Indexes, and $T$ -Indexes

737

The first work that introduced structural indexes was that by Goldman and Widom [23] which introduced a special kind of structural index called the *dataguide*. In their setting, both the data instance and the structural index are rooted graphs. 738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761

The dataguides have the restriction that each node has for each label at most one outgoing edge with that label. The focus is on queries defined by path expressions that are evaluated starting from the root. For certain path expressions and a particular type of dataguide called *strong dataguide*, it is shown that the dataguide is precise. This work was extended by Milo and Suciu in [54] where three types of structural indexes were introduced: the *1-index*, the *2-index*, and the  *$T$ -index*. The *1-index* focuses on the same queries as dataguides, but the merging of nodes is based on an equivalence relation based on bisimulation that follows the edges in reverse, which can be relatively efficiently computed, but still produces a precise index. The *2-index* focuses on queries of the form  $* x_1 P x_2$  where  $*$  is the wildcard path expression that matches any path and  $P$  is an arbitrary path expression. The variables  $x_1$  and  $x_2$  indicate the positions in the path that are returned as a result, so the result is a set of pairs of nodes. A  *$T$ -index* focuses on an even wider class of queries, viz., those of the form  $T_1 x_1 T_2 x_2 \dots T_n x_n$  where each  $T_i$  is either a path expression or a formula expressing a local condition. A different  *$T$ -index* is defined for each query template, where a query template is defined by choosing for certain  $T_i$  a particular path expression, and specifying for others only if they are a path expression or a formula. The *2-index* and  *$T$ -indexes* are not based on merging on single nodes, as in dataguides and *1-indexes*, but rather on tuples of  $n$  nodes or less, but also here, there is an accompanying notion of equivalence and a result that says that these indexes are precise for the queries they are defined for.

#### 2.4.2.2 Localized Notions of (Bi)similarity

762

This work was applied and extended in an XML setting by Kaushik et al. in [39]. Here data instances are modeled as a node-labeled tree with additional `idref` edges that may introduce cycles. The authors observe that *1-indexes* can become very large in practice and that they can be reduced in size if we redefine the equivalence relation for merging by basing it on  $k$ -bisimulation which essentially considers only the local environment of a node within distance  $k$ . This causes the equivalence relation to be coarser, therefore more nodes to be merged, and so produces a smaller index, called an  $A(k)$ -index. However, it also causes the resulting index to be only approximate and no longer precise for the considered query language (namely, regular path expressions). It is shown that this approximation is relatively rare in practice, and 763  
764  
765  
766  
767  
768  
769  
770  
771  
772

that it often can be detected that the result is precise without considering the original  
data instance graph. 773  
774

This work was subsequently extended by Kaushik et al. in [37] for the subset 775  
of XPath known as BPQ, the *Branching Path Queries*. These queries allow the use 776  
of the forward axes `child` and `descendant`, the backward axes `parent` and 777  
`ancestor`, and navigating using the `idref` attributes both forward and backward. 778  
They also allow filtering conditions for intermediate nodes that consist of path 779  
expressions combined with the `and`, `or`, and `not` operators. The basic trick here is 780  
to extend the instance graph with all the reverse edges, i.e., for a `child` edge, 781  
we add a reverse `parent` edge, and for every `idref` edge, we add a reverse 782  
 $\text{idref}^{-1}$  edge. We then can again define a notion of equivalence based on forward 783  
and backward bisimulation, which now looks in both directions, and results in a 784  
precise index. Moreover, we can again reduce the size of the index by considering 785  
only the local environment of a node up to distance  $k$  and sacrificing precision. 786

#### 2.4.2.3 Connections with Query Language Fragments

787

Since for structural indexes there is a trade-off between size and precision, that is a 788  
smaller index is probably less precise, and because efficiency requires that indexes 789  
are as small as possible, it is interesting to determine what the smallest index is that 790  
is still precise for a certain query language. It was, for example, shown by Kaushik 791  
et al. in [37] that for XML and the XPath subset BPQ, the smallest precise structural 792  
index is defined based on forward and backward bisimulation. Similar results were 793  
obtained by Ramanan in [60] where it was shown that for the positive fragment of 794  
BPQ, i.e., without negation, and for TPQ (Tree Pattern Queries), not bisimulation 795  
but simulation defines the smallest precise structural index. Similar results for other 796  
fragments of XPath were obtained by Gyssens et al. in [26] and by Wu et al. in 797  
[84]. The usefulness of such analysis is shown by Fletcher et al. in [19] where an 798  
improved version of  $A(k)$  indexes is proposed, called  $P(k)$  indexes, which 799  
for each  $k$  is shown to be minimal and precise for a certain well-defined practical 800  
XPath fragment. 801

#### 2.4.2.4 Structural Indexes Combined with Other Indexing Techniques

802

Structural indexes can be combined and extended with other indexing techniques. 803  
For example, Kaushik et al. present in [38] an information retrieval system that 804  
retrieves XML documents based on path expression that may contain some position 805  
keywords. In order to efficiently match the keywords, the structural index is 806  
extended with an inverted list that not only maps keywords to element and text 807  
nodes, but also provides a pointer to the index node whose payload contains that 808  
element or text node. This can be seen as an extension to the indexing mechanism 809  
of SIREn, as discussed in Sect. 2.3.2, but now the small trees that represent entities 810  
are generalized to XML documents. 811

Another combination is presented by Arion et al. in [3] where XML documents 812  
 are indexed with a structural index that is essentially a 1-index as presented in 813  
 Sect. 2.4.2 except that it is required to be a tree. Although this may make the index 814  
 larger than necessary, this is compensated by the fact that it allows the use of the 815  
*(pre, post, level)* labeling scheme on the index itself, which allows faster execution 816  
 of path expressions over this tree structure. 817

#### 2.4.2.5 Parameterizable Index Graphs

818

One of the first instances where structural indexes were explicitly used for RDF 819  
 graphs is by Tran and Ladwig [70]. In their work, it is assumed that the RDF graph 820  
 is essentially an edge-labeled graph with the edge labels explicitly distinct from 821  
 the graph nodes. Consequently, in basic graph patterns, it is not allowed to have 822  
 variables in the predicate position of a triple pattern. Although this excludes certain 823  
 typical RDF use cases where data and metadata are queried together, it makes the 824  
 indexing and query optimization problem more similar to that in classical graph 825  
 databases. 826

The fundamental notion is that of *parameterizable index graph*, or PIG, which is 827  
 a structural index based on forward and backward bisimulation, as was introduced 828  
 by Kaushik et al. in [37]. For a certain index, the types of edges that are considered 829  
 for this bisimilarity can be restricted as specified by two sets: one containing the 830  
 edges labels, i.e., predicate names, that are considered for the forward direction and 831  
 the other set those for the backward direction. These sets can be used to tune the 832  
 index for certain workloads. 833

Based on the PIG, three concrete indexing structures are defined: (1) An index 834  
 we will call *PIGidx* that represents the PIG itself and maps predicate names to a 835  
 sorted list of pairs that represent the edges in the PIG associated with that predicate 836  
 name. (2) An index we will call *VPSOIdx* over keyspace  $(v, p, s, o)$  with  $v$  a node 837  
 in PIG, which allow prefix matching, i.e., we can do lookups for  $v$ ,  $vp$  or  $vps$ , and 838  
 returns sorted results. A tuple  $(v, p, s, o)$  is indexed if  $(s, p, o)$  is in the RDF graph 839  
 and  $s$  is in the payload of  $v$ . (3) A similar index we will call *VPOSIdx* over keyspace 840  
 $(v, p, o, s)$ . 841

Given these indexes, the result of a query that is a basic graph pattern can then 842  
 be computed as follows. First we apply the query to the PIG and obtain answers 843  
 in terms of PIG nodes. This can be done by iterating over the triple patterns in 844  
 the basic graph pattern, and for each extend the intermediate result, by equijoining 845  
 it appropriately with the pairs associated by *PIGidx* with the mentioned predicate 846  
 name. For a query with  $n$  triple patterns, the result is a set of tuples of the form 847  
 $T = (v_1, \dots, v_{2n})$  where for  $1 \leq i \leq n$ , each pair  $(v_{2i-1}, v_{2i})$  is a local matching 848  
 for the  $i$ th triple pattern. For such pairs  $(v_{2i-1}, v_{2i})$ , we can define an associated 849  
 set  $L_i$  of pairs at the RDF graph level such that  $L_i$  contains  $(r_{2i}, r_{2i-1})$  iff  $r_{2i}$  and 850  
 $r_{2i-1}$  are in the payload of  $(v_{2i-1}$  and  $v_{2i})$ , respectively, and  $(r_{2i}, p_i, r_{2i-1})$  is in 851

AQ4

the RDF graph, where  $p_i$  the predicate in triple pattern  $i$ . For each such  $T$ , we 852  
 can then construct a set of corresponding tuples of  $2n$  resources in the RDF graph 853  
 by equijoining these  $L_i$  using appropriate equality conditions. Depending on the 854  
 equality conditions and the join order, we can compute each additional join step 855  
 using either VPSOidx or VPOSidx plus some selections. Finally, we take the union 856  
 of all these sets, and project the result such that we have only one column for each 857  
 variable in the query. 858

There are several ways in which these proposed indexes and the query executing 859  
 procedure can be optimized. The PIG can, for example, be made smaller by 860  
 restricting the bisimulation to depth  $k$  as in  $A(k)$  and  $P(k)$  indexes. The query 861  
 execution can be optimized by considering different join orders, and since for certain 862  
 tree-shaped queries PIGidx is in fact a precise index, these can sometimes be pruned 863  
 from a query after the first step of query execution over PIGidx. 864

#### 2.4.2.6 gStore

865

Another RDF datastore that features indexing structures that can be thought of as a 866  
 structural index is gStore, presented by Zou et al. in [86]. In this system, the focus 867  
 is on answering queries with wildcards over disk-based data, and supporting this 868  
 with an index that is easy to maintain. Also here, as for parameterized index graphs, 869  
 the assumption is made that the RDF graph is treated as essentially an edge-labeled 870  
 graph. 871

The key idea of gStore is to assign to each resource not only an integer identifier 872  
 but also a *signature*. The signature is a bit-string that contains some information 873  
 about the triples in which the resource occurs as the subject. It is determined as 874  
 follows. First, we define for each of these relevant triples a bit-string of  $M + N$  875  
 bits, where  $M$  bits are used to encode the predicate and  $N$  bits are used to encode 876  
 the object. The predicate is encoded in  $M$  bits by applying some appropriate hash 877  
 function. If the object is a resource, then it is also mapped by means of a hash 878  
 function to an  $N$  bits number. If the object is a literal, however, then we determine 879  
 the  $N$  bits as follows: first determine the set of 3-grams of the literal, and then apply 880  
 a hash function  $h$  that maps each 3-gram to a number in  $[1\dots N]$ . We set the  $i$ th bit 881  
 to 1 iff there is a 3-gram in the set that is mapped by  $h$  to  $i$ . Given these encodings 882  
 of each triple in  $M + N$  bits, we then combine all these bit-strings into a single 883  
 bit-string of  $M + N$  bits by taking their bitwise OR (remember that all of the triples 884  
 have the same subject  $s$ ). The key insight is if we do the same for variables in a basic 885  
 graph pattern, while mapping variables in the object to  $M + N$  zeros, this gives us 886  
 a quick way to filter out certain impossible matches by checking if the bit-string of 887  
 the variable is subsumed by the bit-string of the resource. 888

The indexing structure built over the signatures of resources is called a VS-tree, a 889  
 vertex signature tree. This is a balanced tree such that each of its leaves corresponds 890  
 to a signature of a resource and the internal nodes correspond to signatures obtained 891

by taking the bitwise OR of the children's signatures. The tree is optimized to 892 efficiently locate a certain signature of a resource when starting from the root. At 893 each level in the tree, we also add edges between tree nodes at the same level. Such 894 an edge indicates the existence of at least one triple between resources associated 895 with the leaves of the subtrees rooted at these tree nodes. More precisely, between 896 index tree nodes  $m_1$  and  $m_2$  at level  $l$ , there will be an edge iff there is a leaf node 897  $m'_1$  below  $m_1$  and a leaf node  $m'_2$  below  $m_2$  such that these correspond to signatures 898 of the resources  $r_1$  and  $r_2$ , respectively, and there is a triple of the form  $(r_1, p, r_2)$ . 899 Moreover, this edge between  $m_1$  and  $m_2$  in the index is labeled with a bit-string 900 of  $M$  bits which is the bitwise OR of the signatures of all predicates that hold 901 among all such  $r_1$  and  $r_2$ . Note that at each higher level of this index, we therefore 902 find a smaller, more abstract, and summarized structural index of the original RDF 903 graph. This allows us to determine all matchings of a basic graph pattern, by starting 904 from the root and moving one level down each time, each step eliminating more 905 impossible matchings and refining those that remain, until we have matchings on the 906 leaves. In the final step, we can translate these to matchings on the actual resources 907 which we can check against the actual RDF graph. 908

## 2.5 Conclusion and Future Work

909

In this chapter, we have provided an up-to-date survey of the rich variety of 910 approaches to storing and indexing very large RDF datasets. As we have seen, this 911 is a very active area of research, with sophisticated contributions from several fields. 912 The survey has been organized by the three main perspectives which unify the many 913 efforts across these fields: relational, entity, and graph-based views on RDF data. 914

We conclude with brief indications for further research. Across all perspectives, a 915 major open issue is the incorporation of schema and ontology reasoning (e.g., RDFS 916 and OWL) in storage and indexing. In particular, there has been relatively little 917 work on the impact of reasoning on disk-based data structures (e.g., see [27, 69]). 918 Furthermore, efficient maintenance of storage and indexing structures as datasets 919 evolve is a challenge for all approaches where many research questions remain 920 open. Possible directions for additional future work in the entity perspective are the 921 investigation of support for richer query languages and integration of entity-centric 922 storage and indexing with techniques from the other two perspectives. Finally, it 923 is clear from the survey that graph-based storage and indexing is currently the least 924 developed perspective. A major additional direction for research here is the develop- 925 ment and study of richer structural indexing techniques and related query processing 926 strategies, following the success of such approaches for semistructured data. 927

**Acknowledgements** The research of FP is supported by an FNRS/FRIA scholarship. The research 928 of SV is supported by the OSCB project funded by the Brussels Capital Region. The research of 929 GF, JH, and YL is supported by the Netherlands Organisation for Scientific Research (NWO). 930

## References

931

- AQ5
1. Abadi, D., Marcus, A., Madden, S., Hollenbach, K.: SW-Store: a vertically partitioned DBMS for semantic web data management. *VLDB J.* **18**, 385–406 (2009) 932  
933
  2. Angles, R., Gutierrez, C.: Survey of graph database models. *ACM Comput. Surv.* **40**, 1:1–1:39 (2008) 934  
935
  3. Arion, A., Bonifati, A., Manolescu, I., Pugliese, A.: Path summaries and path partitioning in modern XML databases. *World Wide Web* **11**, 117–151 (2008) 936  
937
  4. Atre, M., Chaoji, V., Zaki, M.J., Hendler, J.A.: Matrix “bit” loaded: a scalable lightweight join query processor for RDF data. Proceedings of the 19th International Conference on World Wide Web, WWW '10, pp. 41–50. ACM, New York, NY, USA (2010) 938  
939  
940
  5. Bizer, C., Jentzsch, A., Cyganiak, R.: State of the LOD cloud. <http://www4.wiwiiss.fu-berlin.de/lodcloud/state/>. Retrieved July 5, 2011 941  
942
  6. Bönström, V., Hinze, A., Schwerpe, H.: Storing RDF as a graph. Proceedings of the First Conference on Latin American Web Congress, pp. 27–36. IEEE Computer Society, Washington, DC, USA (2003) 943  
944  
945
  7. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: a generic architecture for storing and querying RDF and RDF schema. International semantic web conference, pp. 54–68. Sardinia, Italy (2002) 946  
947  
948
  8. Castillo, R.: RDFMatView: indexing RDF data for SPARQL queries. 9th international semantic web conference (ISWC2010), 2010 949  
950
  9. Center, Q.G.: Bio2RDF. <http://bio2rdf.org/> 951
  10. Chong, E.I., Das, S., Eadon, G., Srinivasan, J.: An efficient SQL-based RDF querying scheme. VLDB, pp. 1216–1227. Trondheim, Norway (2005) 952  
953
  11. Data.gov. <http://www.data.gov> 954
  12. Delbru, R., Campinas, S., Tummarello, G.: Searching web data: an entity retrieval and high-performance indexing model. *Web Semant. Sci. Serv. Agents World Wide Web* (2011, in press) 955
  13. Dong, X., Halevy, A.Y.: Indexing dataspaces, ACM SIGMOD, Beijing, 2007, pp. 43–54 957
  14. Elbassuoni, S., Ramanath, M., Schenkel, R., Weikum, G.: Searching RDF graphs with SPARQL and keywords. *IEEE Data Eng. Bull.* **33**(1), 16–24 (2010) 958  
959
  15. Erling, O.: Towards web scale RDF. SSWS. Karlsruhe, Germany (2008) 960
  16. Erling, O., Mikhailov, I.: RDF support in the virtuoso DBMS. In: Auer S., Bizer C., Müller C., Zhdanova A.V. (eds.) CSSW, *LNI*, vol. 113, pp. 59–68. GI (2007) 961  
962
  17. Fletcher, G.H.L., Beck, P.W.: Scalable indexing of RDF graphs for efficient join processing. CIKM, pp. 1513–1516, Hong Kong, 2009 963  
964
  18. Fletcher, G.H.L., Van Den Bussche, J., Van Gucht, D., Vansumeren, S.: Towards a theory of search queries. *ACM Trans. Database Syst.* **35**, 28:1–28:33 (2010) 965  
966
  19. Fletcher, G.H.L., Van Gucht, D., Wu, Y., Gyssens, M., Brenes, S., Paredaens, J.: A methodology for coupling fragments of XPath with structural indexes for XML documents. *Inform. Syst.* **34**(7), 657–670 (2009) 967  
968  
969
  20. Franklin, M., Halevy, A., Maier, D.: From databases to dataspaces: a new abstraction for information management. *SIGMOD Rec.* **34**, 27–33 (2005) 970  
971
  21. Funche, T., Weinzierl, A., Bry, F.: Labeling RDF graphs for linear time and space querying. In: De Virgilio, R., Giunchiglia, F., Tanca, L. (eds.) Semantic Web Information Management, pp. 309–339. Springer, Berlin, Heidelberg, New York (2009) 972  
973  
974
  22. Goasdoué, F., Karanasos, K., Leblay, J., Manolescu, I.: Rdfviews: a storage tuning wizard for rdf applications. In: Huang, J., Koudas, N., Jones, G.J.F., Wu, X., Collins-Thompson, K., An, A. (eds.) CIKM, pp. 1947–1948. ACM, New York (2010) 975  
976  
977
  23. Goldman, R., Widom, J.: DataGuides: enabling query formulation and optimization in semistructured databases. VLDB, pp. 436–445, Athens, Greece, 1997 978  
979
  24. Gou, G., Chirkova, R.: Efficiently querying large XML data repositories: a survey. *IEEE Trans. Knowl. Data Eng.* **19**(10), 1381–1403 (2007) 980  
981
- AQ6

25. Groppe, S., Groppe, J., Linnemann, V.: Using an index of precomputed joins in order to speed up SPARQL processing. ICEIS, pp. 13–20, Funchal, Madeira, Portugal, 2007 982  
983
26. Gyssens, M., Paredaens, J., Van Gucht, D., Fletcher, G.H.L.: Structural characterizations of the semantics of XPath as navigation tool on a document. ACM PODS, pp. 318–327, Chicago, 984  
985  
2006 986
27. Haffmans, W.: A study of efficient RDFS entailment in external memory. Master's thesis, 987  
Eindhoven University of Technology (2011) 988
28. Halevy, A.Y.: Answering queries using views: a survey. VLDB J. **10**(4), 270–294 (2001) 989
29. Halevy, A.Y., Franklin, M.J., Maier, D.: Principles of dataspace systems. PODS, pp. 1–9, 990  
Chicago, 2006 991
30. Harris, S.: SPARQL query processing with conventional relational database systems. Web 992  
Inform. Syst. Eng. WISE 2005 Workshops **3807**, 235–244 (2005) 993
31. Harris, S., Gibbins, N.: 3store: efficient bulk RDF storage. PSSS1, Proceedings of the First 994  
International Workshop on Practical and Scalable Semantic Systems, pp. 1–15, Sanibel Island, 995  
Florida, 2003 996
32. Harth, A., Decker, S.: Optimized index structures for querying RDF from the web. IEEE LA- 997  
WEB, pp. 71–80, Buenos Aires, Argentina (2005) 998
33. Harth, A., Umbrich, J., Hogan, A., Decker, S.: YARS2: a federated repository for querying 999  
graph structured data from the web. ISWC. Busan, Korea (2007) 1000
34. Hayes, P.: RDF Semantics. W3C Recommendation (2004) 1001
35. Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space. Morgan and 1002  
Claypool, San Francisco (2011) 1003
36. Hertel, A., Broekstra, J., Stuckenschmidt, H.: RDF storage and retrieval systems. In: Staab, 1004  
S., Rudi Studer, D. (eds.) Handbook on Ontologies, International Handbooks on Information 1005  
Systems, pp. 489–508. Springer, Berlin, Heidelberg (2009) 1006
37. Kaushik, R., Bohannon, P., Naughton, J.F., Korth, H.F.: Covering indexes for branching path 1007  
queries. ACM SIGMOD, pp. 133–144. Madison, WI (2002) 1008
38. Kaushik, R., Krishnamurthy, R., Naughton, J.F., Ramakrishnan, R.: On the integration of 1009  
structure indexes and inverted lists. ACM SIGMOD, pp. 779–790. Paris (2004) 1010
39. Kaushik, R., Shenoy, P., Bohannon, P., Gudes, E.: Exploiting local similarity for indexing paths 1011  
in graph-structured data. IEEE ICDE, pp. 129–140. San Jose, CA (2002) 1012
40. Klyne, G., Carroll, J.J.: Resource description framework (RDF): concepts and abstract syntax. 1013  
W3C Recommendation (2004) 1014
41. Kolas, D., Emmons, I., Dean, M.: Efficient linked-list RDF indexing in parliament. In: 1015  
Fokoue A., Guo Y., Liebig T. (eds.) Proceedings of the 5th International Workshop on 1016  
Scalable Semantic Web Knowledge Base Systems (SSWS2009), CEUR, vol. 517, pp. 17–32. 1017  
Washington DC, USA (2009) 1018
42. Kolas, D., Self, T.: Spatially-augmented knowledgebase. In: Aberer, K., Choi, K.S., Noy, N.F., 1019  
Allemand, D., Lee, K.I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., 1020  
Schreiber, G., Cudré-Mauroux, P. (eds.) ISWC/ASWC, Lecture Notes in Computer Science, 1021  
vol. 4825, pp. 792–801. Springer, Berlin Heidelberg (2007) 1022
43. Konstantinou, N., Spanos, D.E., Mitrou, N.: Ontology and database mapping: a survey of 1023  
current implementations and future directions. J. Web Eng. **7**(1), 1–24 (2008) 1024
44. Ladwig, G., Tran, T.: Combining query translation with query answering for efficient keyword 1025  
search. ESWC, pp. 288–303. Crete (2010) 1026
45. Lee, J., Pham, M.D., Lee, J., Han, W.S., Cho, H., Yu, H., Lee, J.H.: Processing SPARQL queries 1027  
with regular expressions in RDF databases. Proceedings of the ACM Fourth International 1028  
Workshop on Data and Text Mining in Biomedical Informatics, DTMBIO '10, pp. 23–30. 1029  
ACM, New York, NY, USA (2010) 1030
46. Levandoski, J.J., Mokbel, M.F.: RDF data-centric storage. ICWS, pp. 911–918. IEEE (2009) 1031
47. Li, Q., Moon, B.: Indexing and querying xml data for regular path expressions. Proceedings of 1032  
the 27th International Conference on very Large data bases, VLDB '01, pp. 361–370. Morgan 1033  
Kaufmann Publishers Inc., San Francisco, CA, USA (2001) 1034

48. Lim, L., Wang, M., Padmanabhan, S., Vitter, J.S., Agarwal, R.: Dynamic maintenance of web indexes using landmarks. Proceedings of the 12th International Conference on World Wide Web, WWW '03, pp. 102–111. ACM, New York, NY, USA (2003) 1035  
1036  
1037
49. Ma, L., Su, Z., Pan, Y., Zhang, L., Liu, T.: RStar: an RDF storage and query system for enterprise resource management. ACM CIKM, pp. 484–491. Washington, D.C. (2004) 1038  
1039
50. del Mar Roldán García, M., Montes, J.F.A.: A survey on disk oriented querying and reasoning on the semantic web. IEEE ICDE workshop SWDB. Atlanta (2006) 1040  
1041
51. Matono, A., Amagasa, T., Yoshikawa, M., Uemura, S.: An indexing scheme for RDF and RDF schema based on suffix arrays. SWDB, pp. 151–168. Berlin (2003) 1042  
1043
52. Matono, A., Amagasa, T., Yoshikawa, M., Uemura, S.: A path-based relational RDF database. ADC, pp. 95–103. Newcastle, Australia (2005) 1044  
1045
53. McGlothlin, J.P., Khan, L.R.: Rdfkb: efficient support for rdf inference queries and knowledge management. In: Desai B.C., Saecà D., Greco S. (eds.) IDEAS, ACM international conference proceeding series, pp. 259–266. ACM (2009) 1046  
1047  
1048
54. Milo, T., Suciu, D.: Index structures for path expressions. ICDT, pp. 277–295. Jerusalem (1999) 1049
55. Minack, E., Sauermann, L., Grimnes, G., Fluit, C.: The sesame lucenesail: Rdf queries with full-text search. Technical report, NEPOMUK Consortium, (2008) 1050  
1051
56. Neumann, T., Weikum, G.: RDF-3X: a RISC-style engine for RDF. VLDB. Auckland, New Zealand (2008) 1052  
1053
57. Neumann, T., Weikum, G.: x-RDF-3X: fast querying, high update rates, and consistency for RDF databases. Proc. VLDB Endow. **3**, 256–263 (2010) 1054  
1055
58. Oren, E., Kotoulas, S., Anadiotis, G., Siebes, R., ten Teije, A., van Harmelen, F.: Marvin: distributed reasoning over large-scale semantic web data. J. Web Semant. **7**(4), 305–316 (2009) 1056  
1057
59. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Recommendation (2008) 1058  
1059
60. Ramanan, P.: Covering indexes for XML queries: bisimulation – simulation – negation. Proceedings of the 29th International Conference on very Large Data Bases – Volume 29, VLDB '2003, pp. 165–176. VLDB Endowment (2003) 1060  
1061  
1062
61. Rao, P., Moon, B.: Sequencing XML data and query twigs for fast pattern matching. ACM Trans. Database Syst. **31**, 299–345 (2006) 1063  
1064
62. Rohloff, K., Dean, M., Emmons, I., Ryder, D., Sumner, J.: An evaluation of triple-store technologies for large data stores. OTM 2007 Workshop SSWS, pp. 1105–1114. Vilamoura, Portugal (2007) 1065  
1066  
1067
63. Rohloff, K., Schantz, R.E.: Clause-iteration with mapreduce to scalably query datagraphs in the shard graph-store. Proceedings of the Fourth International Workshop on Data-Intensive Distributed Computing, DIDC '11, pp. 35–44. ACM, New York, NY, USA (2011) 1068  
1069  
1070
64. Sakr, S., Al-Naymat, G.: Relational processing of RDF queries: a survey. ACM SIGMOD Record **38**, 23–28 (2009). URL <http://www.sigmod.org/publications/sigmod-record/0912> 1071  
1072
65. Schmidt, M., Hornung, T., Küchlin, N., Lausen, G., Pinkel, C.: An experimental comparison of RDF data management approaches in a SPARQL benchmark scenario. ISWC, pp. 82–97. Karlsruhe (2008) 1073  
1074  
1075
66. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP<sup>2</sup>Bench: a SPARQL performance benchmark. IEEE ICDE. Shanghai (2009) 1076  
1077
67. Sidiropoulos, L., Goncalves, R., Kersten, M., Nes, N., Manegold, S.: Column-store support for RDF data management: not all swans are white. Proc. VLDB Endow. **1**, 1553–1563 (2008) 1078  
1079
68. Sintek, M., Kiesel, M.: RDFBroker: a signature-based high-performance RDF store. ESWC, pp. 363–377. Budva, Montenegro (2006) 1080  
1081
69. Theoharis, Y., Christophides, V., Karvounarakis, G.: Benchmarking database representations of RDF/S stores. In: Gil Y., Motta E., Benjamins V.R., Musen M.A. (eds.) International semantic web conference. Lecture Notes in Computer Science, vol. 3729, pp. 685–701. Springer, Berlin, Heidelberg, New York (2005) 1082  
1083  
1084  
1085
70. Tran, T., Ladwig, G.: Structure index for RDF data. Workshop on Semantic Data Management (SemData@ VLDB) (2010) 1086  
1087

71. Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. ICDE, pp. 405–416. Shanghai (2009) 1088  
1089
72. Tummarello, G., Cyganiak, R., Catasta, M., Danielczyk, S., Delbru, R., Decker, S.: Sig.ma: live views on the web of data. Web Semant. Sci. Serv. Agents World Wide Web **8**(4), 355–364 (2010). DOI 10.1016/j.websem.2010.08.003 1090  
1091  
1092
73. Udrea, O., Pugliese, A., Subrahmanian, V.S.: GRIN: a graph based RDF index. AAAI, pp. 1465–1470. Vancouver, B.C. (2007) 1093  
1094
74. Valduriez, P.: Join indices. ACM Trans. Database Syst. **12**, 218–246 (1987) 1095
75. W3C SWEO Community Project: Linking open data. <http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData> 1096  
1097
76. Wang, H., Liu, Q., Penin, T., Fu, L., Zhang, L., Tran, T., Yu, Y., Pan, Y.: Semplore: a scalable IR approach to search the web of data. Web Semant. Sci. Serv. Agents World Wide Web **7**(3), 177–188 (2009) 1098  
1099  
1100
77. Wang, X., Wang, S., Pufeng, D., Zhiyong, F.: Path summaries and path partitioning in modern XML databases. Int. J. Modern Edu. Comput. Sci. **3**, 55–61 (2011) 1101  
1102
78. Weikum, G., Theobald, M.: From information to knowledge: harvesting entities and relationships from web sources. PODS, pp. 65–76. Indianapolis (2010) 1103  
1104
79. Weiss, C., Karras, P., Bernstein, A.: Hexastore: sextuple indexing for semantic web data management. VLDB, Auckland, New Zealand (2008) 1105  
1106
80. Wilkinson, K.: Jena property table implementation. SSWS, pp. 35–46. Athens, Georgia, USA (2006) 1107  
1108
81. Witten, I., Moffat, A., Bell, T.: Managing Gigabytes: Compressing and Indexing Documents and Images. Morgan Kaufmann, Los Altos, CA (1999) 1109  
1110
82. Wood, D., Gearon, P., Adams, T.: Kowari: a platform for semantic web storage and analysis. XTech. Amsterdam (2005) 1111  
1112
83. Wu, G., Li, J.: Managing large scale native RDF semantic repository from the graph model perspective. ACM SIGMOD Workshop IDAR, pp. 85–86. Beijing (2007) 1113  
1114
84. Wu, Y., Gucht, D.V., Gyssens, M., Paredaens, J.: A study of a positive fragment of path queries: Expressiveness, normal form and minimization. Comput. J. **54**(7), 1091–1118 (2011) 1115  
1116
85. Yang, M., Wu, G.: Caching intermediate result of sparql queries. In: Srinivasan, S., Ramamirtham, K., Kumar, A., Ravindra, M.P., Bertino, E., Kumar, R. (eds.) WWW (Companion Volume), pp. 159–160. ACM, New York (Los Angeles, CA) (2011) 1117  
1118  
1119
86. Zou, L., Mo, J., Chen, L., Özsu, M.T., Zhao, D.: gStore: answering SPARQL queries via subgraph matching. Proc. VLDB Endowment **4**(8), 482–493 (2011) 1120  
1121

AUTHOR QUERIES

- AQ1. Kindly check the corresponding author.
- AQ2. Please confirm the changes in the sentence “They remark in particular...”.
- AQ3. Please check the sentence “The simplest way to...”.
- AQ4. In the inline list here, please check if ‘we will call’ can be removed from the list items.
- AQ5. Kindly provide the accessed date for [9,11,75].
- AQ6. Kindly update Ref. [12].

UNCORRECTED PROOF

# Chapter 3

# Designing Exploratory Search Applications

## upon Web Data Sources

Marco Brambilla and Stefano Ceri

### 3.1 Introduction

Search is the preferred method to access information in today’s computing systems. 6  
The Web, accessed through search engines, is universally recognized as the source 7  
for answering users’ information needs. However, offering a link to a Web page 8  
does not cover all information needs. Even simple problems, such as “Which 9  
theater offers an at least three-stars action movie in London close to a good Italian 10  
restaurant,” can only be solved by searching the Web multiple times, e.g., by 11  
extracting a list of the recent action movies filtered by ranking, then looking for 12  
movie theaters, then looking for Italian restaurants close to them. While search 13  
engines hint to useful information, the user’s brain is the fundamental platform for 14  
information integration. 15

An important trend is the availability of new, specialized data sources—the so- 16  
called “long tail” of the Web of data. Such carefully collected and curated data 17  
sources can be much more valuable than information currently available in Web 18  
pages; however, many sources remain hidden or insulated, in the lack of software 19  
solutions for bringing them to surface and making them usable in the search context. 20  
A new class of tailor-made systems, designed to satisfy the needs of users with 21  
specific aims, will support the publishing and integration of data sources for vertical 22  
domains; the user will be able to select sources based on individual or collective 23  
trust, and systems will be able to route queries to such sources and to provide easy- 24  
to-use interfaces for combining them within search strategies, at the same time, 25  
rewarding the data source owners for each contribution to effective search. Efforts 26  
such as Google’s Fusion Tables show that the technology for bringing hidden data 27  
sources to surface is feasible. 28

---

AQ1 M. Brambilla (✉) · S. Ceri  
Dipartimento di Elettronica e Informazione, Politecnico di Milano, P.zza L. Da Vinci, 32. I-20133  
Milano, Italy  
e-mail: [marco.brambilla@polimi.it](mailto:marco.brambilla@polimi.it); [stefano.ceri@polimi.it](mailto:stefano.ceri@polimi.it)

In this chapter, we focus on building complex search applications upon structured Web data, by providing a set of software engineering guidelines and tools to the designer. We assume that the search services available on the Web have been identified and registered within a search service integration platform, and based on that, we allow for the definition of vertical applications through conceptual definition of the domains of interest. In particular, we define a pattern-based approach for defining the structure of Web entities, for defining their integration, and for specifying information seeking strategies upon them. We focus on *exploratory queries*, that is, queries where, given the current context of interaction, the user is able to follow links to connected concepts, thus adding a new query fragment or rolling back to a previous result combination. Exploratory queries are by nature incremental. Our proposal for addressing exploratory search upon multidomain, structured Web data sources, is presented in the Liquid Query paradigm [5], which allows users to interact with the search computing result by asking the system to produce “more result combinations,” or “more results from a specific service,” or “performing an expansion of the result” by adding a subquery which was already planned while configuring the query.

The technological support for our work is provided by the search computing (SeCo) framework [14, 15], which provides a general purpose Web integration platform and search engine upon distinct sources. The main purpose of our framework is lowering the complexity of building search applications. The impact of this work can be potentially high: when the complexity threshold of building structured entity search applications will be lowered, a variety of new market sectors will become more profitable and accessible. In several scenarios, search-enabled Web access will grow in interest and value when SMEs or local businesses will see the opportunity of building search applications tailored to their market niche or sale region.

The chapter is organized as follows: Sect. 3.2 discusses the related work; Sect. 3.3 defines the basic concepts for describing entities and their connections on the Web; Sect. 3.4 defines the design patterns for building Web applications, presenting several examples; Sect. 3.5 shows how some examples have been already deployed as Web applications within SeCo; Sect. 3.6 presents the tool platform for allowing query execution, service registration, and application deployment; and Sect. 3.7 concludes.

## 3.2 State of the Art

The design of novel search systems and interfaces is backed by several studies aimed at understanding users' search behavior on the Web. After the seminal work by Broder [13], other studies have investigated search behaviors and effectiveness of result page composition [16] by analyzing search engine logs. A review of approaches to search log data mining and Web search investigation is in [2].

A specific class of studies is devoted to exploratory search, where the user's intent is primarily to learn more on a topic of interest [20, 25]. Such information seeking behavior challenges the search engine interface, because it requires support to all the stages of information acquisition, from the initial formulation of the area of interest, to the discovery of the most relevant and authoritative sources, to the establishment of relationships among the relevant information elements. A number of techniques have been proposed to support exploratory search, and user studies have been conducted to understand the effectiveness of the various approaches (e.g., [19]), including analyses of the involvement of the user in the information extraction process [21].

Linked Data (LD) and other open or proprietary data are made available through Web APIs (e.g., see Google Places API and Google Fusion Tables [17]) and/or search-specific languages (e.g., see the Yahoo query language (YQL) framework [26]). Methods, models, and tools are being devised for efficiently designing and deploying applications upon such new data services and data access APIs. An important aspect of LD is their use of universal references for data linking; this aspect raises the hopes of solving the data-matching problem, which has so far limited the practical applicability of data integration methods.

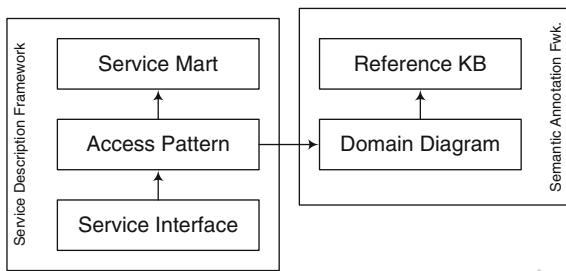
However, data access and linking so far has not been concerned with data search. The efficient support of such data services requires mastering both data and control dependencies, and strategy optimization must consider rank aggregation, optimal result paging, and so on [9]. When data sources must be joined, the join operation must take into account ranking [8]; join can either be based on exact methods, according to the rank-join theory, or on approximate methods, that favor the speed of result production [18]. Data integration strategies should aim at obtaining a suitable number of results (e.g., neither too few, nor too many). Normally, a computation should not be set up so as to exhaust a searchable data source, as the user is rarely interested to inspect all of them [10]. The ultimate controller is the user, who sees service results or their compositions and can halt their production.

The search system Kosmix [23] already uses such an approach for describing data sources that is sufficient for selecting relevant sources and driving simple queries to them, but it currently does not support the combination of data sources through operations.

### 3.3 Structured Entity Registration and Integration on the Web

An increasing number of data sets are becoming available on the Web as (semi)structured data instead of user-consumable pages. Linked Data plays a central role in this, thanks to initiatives such as W3C Linked Open Data (LOD) community project, which are fostering LD best practice adoption [4]. However, several other kinds of (nonlinked) data sources can be available on the Web.

**Fig. 3.1** Overview of the description levels for a search service



We envision the integration and search upon generic Web data sources through their registration and semiautomatic tagging. Tags can be extracted from general ontologies (such as YAGO) so as to build an abstract view of the sources that can be helpful in routing queries to them [24].

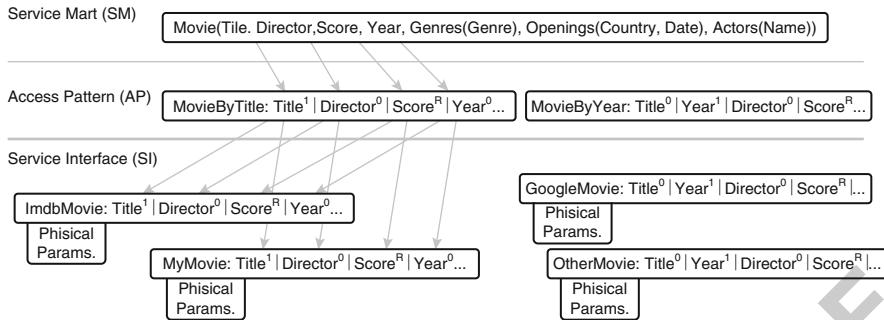
The feasibility of the approach is demonstrated by the search computing framework (<http://www.search-computing.org>). In this framework, services are registered and modeled according to the different layers described in Fig. 3.1. During registration, each data service becomes known in terms of: the entities that it describes (conceptual view), its access pattern (logical view), and the call interface, with a variety of QoS parameters (physical view); these three views compose the service description framework (SDF) at different levels of abstractions [22]. Access pattern information from all the services is used to create the domain diagram (DD), referring in turn to one or more knowledge bases (KB); these views describe the semantic annotation framework (SAF).

At the conceptual level, the definition of a service mart includes the object's name and the collection of the object's attributes. All attributes are typed; attributes can be atomic (single valued) or part of a repeating group (multivalued). Service marts are specific data patterns; their regular organization helps structuring search computing applications, in a way very similar to the so-called "data marts" used in the context of data warehouses. Service marts are instrumental in supporting the notion of "Web of objects" [1] that is gaining popularity as a new way to think of the Web, going beyond the unstructured organization of Web pages.

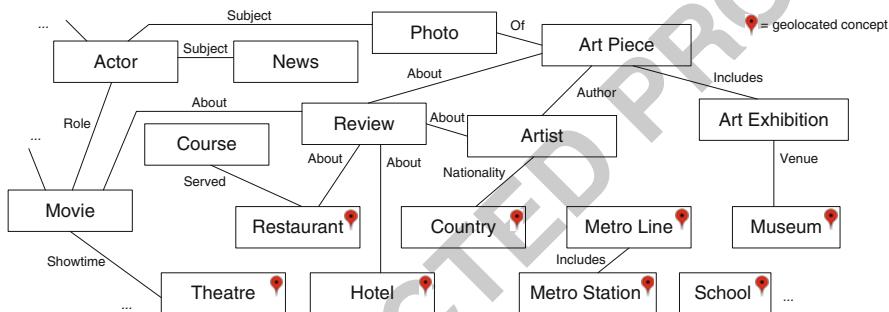
At the logical level, service marts are associated with one or more access patterns representing the signatures of service calls. Access patterns contain a subset of the service mart's attributes tagged with I (input), O (output), or R (ranking). Ranking attributes are essential to characterize the nature of the service, as we expect most services to respond to search queries by producing itemized results' lists in rank order; moreover, when rank attributes are explicitly presented, they are used by the search computing system to associate a global ranking to query results, and then present them to users in global rank order.

At the physical level, each access pattern may be mapped to several service implementations, called service interfaces. Service interfaces are registered by providing a service identifier and the names of physical parameters, which are mapped to the access pattern's attributes. Additional descriptions include details

AQ3



**Fig. 3.2** Example of descriptions for accessing movie information through the service mart, access pattern, and service interface layers



**Fig. 3.3** Example of service mart diagram with connected concepts

about the Web service endpoint and the protocol to be used for its invocation, 143  
together with basic statistics on the service behavior describing the quality of service 144  
(QoS). 145

Figure 3.2 provides a concrete example of some registered services, all referring 146  
to the concept of movie. Movie is registered as a service mart, together with 147  
two associated access patterns and service interfaces, with the respective attribute 148  
mappings (mappings are shown only for the first access pattern for brevity). 149

Figure 3.3 is a representation at the service mart level of a set of registered 150  
concepts, covering information from different fields. Figure 3.3 also shows some 151  
of the relationships between concepts, which are drawn when the concepts share 152  
parameters, enabling navigation from one of them (in output) to another (in input). 153  
Indeed, more navigations are possible, e.g. when they share output parameters 154  
that can be compared (joined), or when outputs from two or more concepts 155  
enable filling the input for a third concept. Moreover, concepts which are labeled 156  
by a localization symbol may be connected by geographic proximity. Thus, the 157  
representation of Fig. 3.3 only hints to the complexity of interconnecting services, 158  
which is indeed fully captured by the data dictionary mentioned in Fig. 3.1; data 159  
dictionary connections, called connection patterns, are drawn at the logical level as 160

they depend on attribute tags. Navigating and exploring information based on this view is not convenient for end users, which typically require more focused and well-structured interactions and interfaces. Therefore, in the next section, we discuss how to design applications whose complexity is suited to user interaction, where each application focuses on few concepts, and highlights some concept relationships for exploring those concepts in a manageable way.

### 3.4 Design of Web Applications

A Web application is built around a specific search process that a user wants to perform. Examples of processes range from very simple ones, such as deciding where to spend an evening in a city by choosing restaurants and other amenities, such as concerts, movies, or art festivals, to more complex and long-term ones, such as exploring options for job offers and house offerings. Such tasks may complete in one interaction or instead require many intermediate steps, where a user can progressively select subsets of the results, browse them, and possibly backtrack decisions. Examples of such applications in the search computing context have been demonstrated [5].

Designing a Web application requires the selection of one specific *goal* for each application; examples of goals are “planning an evening in SF,” or “planning long weekend options from Chicago,” or “selecting candidate job offers for software programmers in the bay area,” or “browsing postdoc offers in cloud computing.” The *processes* for achieving such goals may be classified as either *short-term* (serviced by a session whose span is accounted in seconds or at most minutes) or *long-lived* (spanning over days, in which case a search session should be suspended and then resumed—and partial results may be subject to a verification by repeating queries).

Processes are divided into smaller *tasks*; as design method, we associate each task with exactly one interaction with the search computing system. Each task consists of getting input parameters, issuing a query over services, presenting results to the user, allowing them to browse the instance and perform operations upon them (e.g., selecting the most interesting ones), and then decide the next task or exit/save the session—therefore, in each task execution, we expect an interaction with one or more services. While short-term processes may require few task interactions (possibly just one), long-lived processes may require many tasks.

Tasks are chosen according to a *workflow*, which presents legal sequences of tasks or choices of tasks; search computing natively supports search histories, and therefore, a possible user’s choice of next task is always backtracking an arbitrary number of tasks in the history. This allows workflows to be quite simple, consisting either of free choices among tasks, or of hierarchies where the execution of one task opens up the possibility of executing lower-level tasks; a hierarchy often degenerates to a simple sequence when one task is followed by just one task.

After every task execution, the search computing system builds the “new” *task result* by joining the “old” result with the results produced by the task’s

query; therefore, the sequential execution of two tasks in the workflow requires 202  
the existence of a connection pattern between the corresponding services. This is 203  
obviously a strong constraint in building applications, but it yields many advantages, 204  
such as giving a precise semantics to the “current query” and its results throughout 205  
a session, allowing results to be globally ordered according to a “current goal 206  
function” (typically a weighted function or ranks produced by services), and thus 207  
allowing the presentation of the top items and/or the use of diversification for 208  
producing interesting items. Such computations are supported by Panta Rhei, the 209  
search computing query execution engine [11]. 210

The method presented above consists in choosing among well-identified *patterns* 211  
for task composition, and then verifying that such patterns are supported by access 212  
and connection patterns of tasks. Such patterns are: the fully connected network, 213  
the hierarchy (either forming a single star or a snowflake), and the sequence, where 214  
tasks can either be simple (mapped to one service) or composite (mapped to many 215  
services). In the following, we present several designs of Web applications based on 216  
the above patterns. 217

### 3.4.1 Night Planner

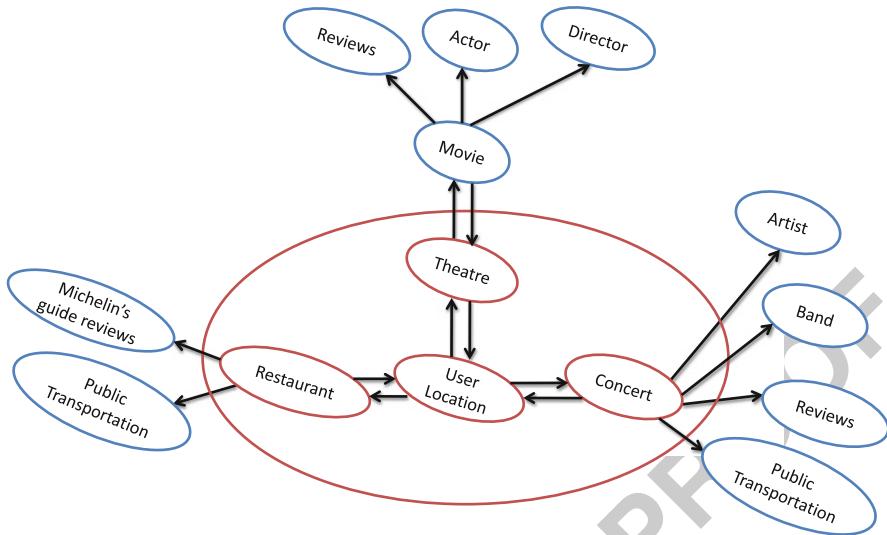
218

A night planner is a short-term Web application presenting several geolocalized 219  
services, describing restaurants, shows, movies, family events, music concerts, and 220  
the like. The workflow is free; therefore, any service can be chosen after any other 221  
service, using the fully connected pattern; a *night plan* is just the combination 222  
of several such services, representing a dinner option, then a music concert, and 223  
so on. Selected restaurants are ranked by distance from the user and possibly 224  
by their score; when the concert service is queried, it produces combinations of 225  
restaurants and shows, and their ranking is based on the new distance between the 226  
restaurant and the concert hall, the previous distance from the user location, and 227  
the scores of restaurants and concerts; diversification allows seeing appropriately 228  
chosen combinations of restaurants and concerts. The planner may be augmented 229  
hierarchically with services showing reviews of each plan ingredient, and/or some 230  
options for public transportation from home or the closest parking; these selections 231  
only make sense in the context of few specific alternatives of the current solution. 232  
The resulting application design is shown in Fig. 3.4. 233

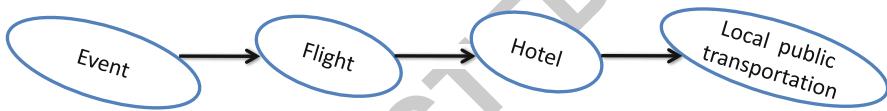
### 3.4.2 Weekend Browser

234

A weekend browser is a short-term Web application presenting to users the events 235  
which are occurring in one or more selected cities of interest. The application 236  
is sequential, because the user enters the application by browsing events; but 237  
once he is considering a particular location, he is offered additional services for 238



**Fig. 3.4** Night Planner application design



**Fig. 3.5** Weekend browser

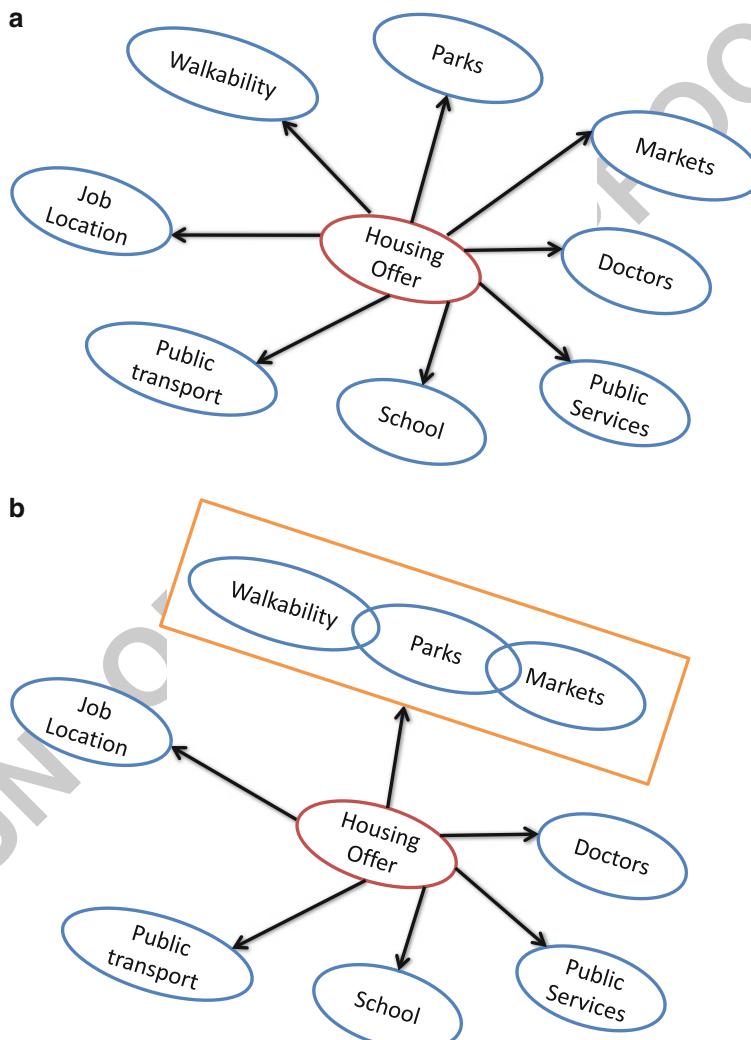
completing the weekend plan by adding transportation to the location (flights), then 239  
hotels in the location, then transportation from the airport to the hotel and to the 240  
chosen events. The sequence is defined by the application designer, who establishes 241  
that attending the event is the primary motivation for using this application, then 242  
checking the existence of a suitable transportation, and finally completing the plan 243  
by adding the hotel choice and local transports to it. Several instances of this 244  
application can be further specialized by event types (e.g., be associated with a sport 245  
team allowing fans to follow road games of the team, or be specialized on classic 246  
concerts within Europe). The resulting application design is shown in Fig. 3.5. 247

### 3.4.3 Real-Estate Browser

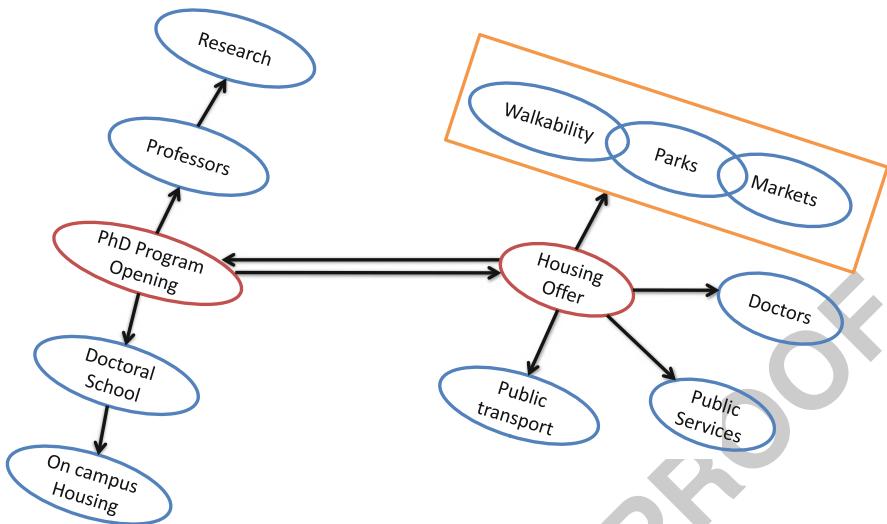
248

A real-estate browser is a long-lived, hierarchical application. It is centered around a 249  
real-estate service or service integrator presenting offers (for buy or rent) of various 250  
houses, and in addition, it has services which describe the house district or proximity 251  
in terms of: distance from work; availability of public transport from work; vicinity 252  
to schools of given grade, to supermarkets, to green and parks, and to hospitals and 253

specialist doctor's clinics; and the like. These services are ancillary to the primary service, as each user may be interested in a subset of them. In a typical interaction, a user may select some house offers (e.g., based on their area, cost, and bedrooms) and evaluate them according to some search dimensions (e.g., distance from work, distance from doctor's clinics for some disease). The designer may simplify the interaction by combining several services into one query (e.g., walkability and vicinity to markets and parks). The two variants of the application are shown in Fig. 3.6a, b.



**Fig. 3.6** (a) Real-estate offer browser, with (b) a variant showing several services explored as a single task (and a single query to several data sources)



**Fig. 3.7** Job-house combination browser specialized for PhD applications

### 3.4.4 Job-House Combination Browser

262

A work-job browser is a long-lived, hierarchical application where two hierarchical 263 roots, one centered on work offers and one on house offers, are present; depending 264 on the choice of the designer, this may be considered as a hierarchy (work offers 265 come first) or a simple free interaction (work and house offers equally relevant). 266 While the house star is a replica of the one presented in Sect. 3.4.3, the job star is 267 centered on the notion of job offer. Specialized versions of such a service may exist, 268 e.g., Fig. 3.7 shows the application as designed for applicants to PhD programs, 269 where openings are linked to doctoral schools, then to their professors, then to their 270 research programs, and an on-campus housing may be directly linked to the doctoral 271 school. 272

## 3.5 Aspects of a Real-Estate Exploratory Browser

273

Web applications which use an exploratory approach can be built in a variety of 274 ways; in SeCo, we designed an orchestrator-based approach where each exploratory 275 step is based upon a cycle of: selection of the next service to be invoked, 276 provisioning of input parameters, query execution, data display, and data processing. 277 Data processing, in turn, may consist of the request for more data, or the selection 278 of some results for further investigations, or of projection/aggregation/reordering 279 of current results, before entering the next step. The orchestrator allows recovering 280

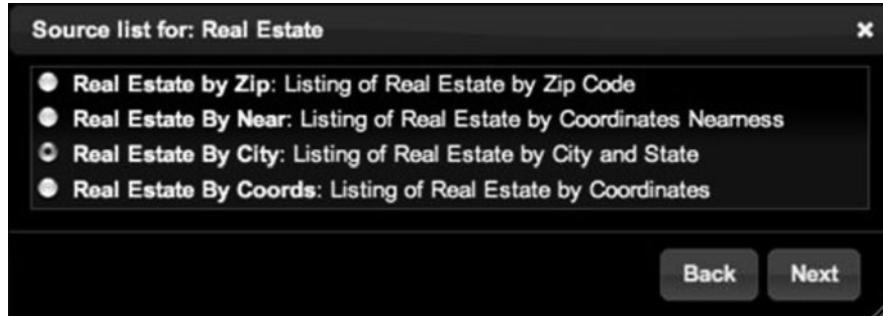


Fig. 3.8 Service selection form

previous steps, and in particular, this may result in backtracking one or more steps, 281  
which is typical of explorations. 282

We also developed generic UIs which depend on the specific data type results. 283  
Generic data are represented through “tabular” or “atom” views; geolocalized data 284  
are placed on maps—together with all Linked data which can be associated to geo- 285  
locations through direct mappings; quantitative data can be plotted on XY diagrams, 286  
histograms, timelines, and so on. In [3], we demonstrated aspects of the job-house 287  
combination browser, by showing how the generic orchestrator and UIs adapt to that 288  
particular application; here we focus on the real-estate part. 289

Figure 3.8 shows a form-based UI for selecting services at the beginning of 290  
one step, based upon the specific available information (a ZIP code, city and state, 291  
coordinates, nearness to coordinates.) 292

The user chooses the third option based on his current location (Stagg Street in 293  
Brooklyn, NY; coordinates are automatically obtained from mobile devices), and is 294  
offered a number of alternative house offerings, displayed in Fig. 3.9. Each offer is 295  
associated with a ranking which depends on the distance and also from the closeness 296  
of the offer with specified search criteria; based on the selected service and API, the 297  
user is prompted with an additional form for specifying alternatives between buy 298  
and rent, number of bedrooms, price range, etc., to better focus the search (in this 299  
particular example, we used zillow, www.zillow.com.) 300

Next, several other services can be inspected which give information about 301  
nearby services, e.g. schools, walk score, hotels, doctors, theaters, events, car 302  
rentals, census, news, job offers in proximity, restaurants; these alternatives are 303  
shown in the form of Fig. 3.10. Note that the richness of options here is the 304  
novelty, compared to a fixed set of options that is typically provided by a real- 305  
estate integrator. Some may be available due to choices of specific users, who made 306  
previous use of such services and tagged them to be always available whenever they 307  
search is “by coordinate,” as in the current situation. 308

In the continuation of the session, we assume the user to check for medical 309  
specialists, and specifically for cardiologists, in the neighborhood of two house 310  
offers that were selected at the previous step; the result is then a list of combinations 311

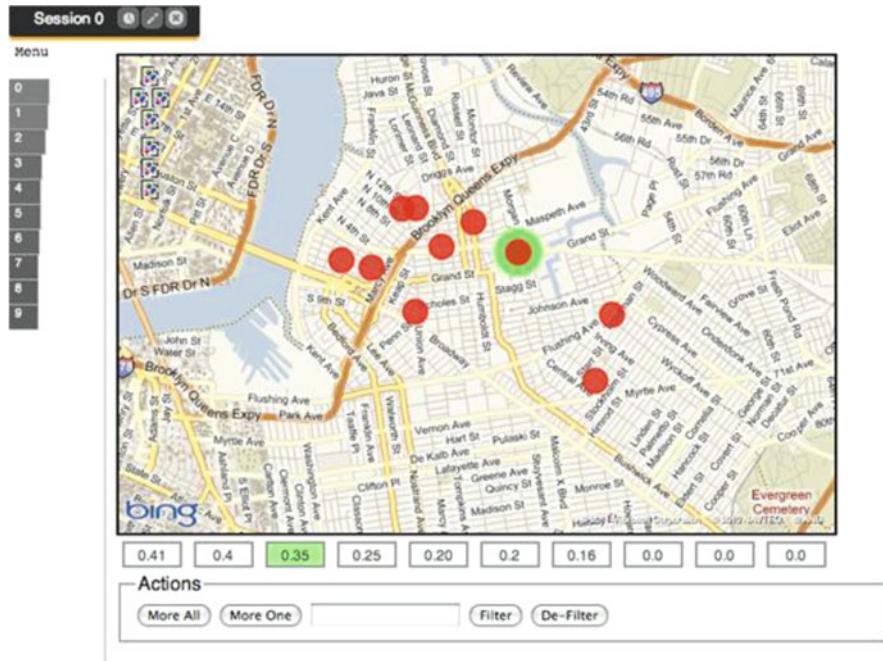


Fig. 3.9 House offers in proximity of a given location

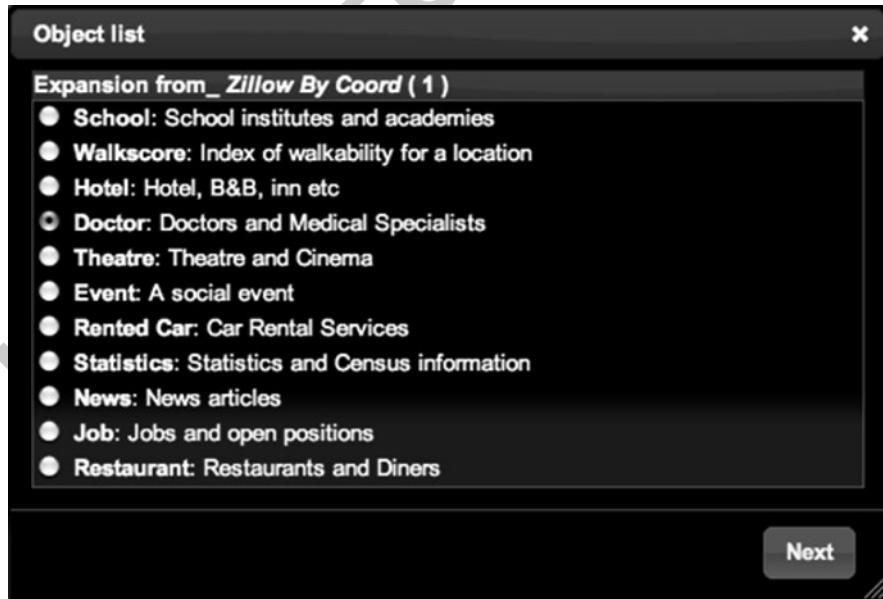


Fig. 3.10 Service exploration in “proximity” of house offer locations

of interconnected pairs, where each pair represents housing (red) and doctor's offices (green), and certain pairs are ranked better than others by taking into account the geographic distance between locations, the doctor's office reputation, and the original ranking of the two house offers. Figure 3.11a, b show the display of offers on the map and in the atom view, which is selectively displaying house offers' usecode, price and bedrooms, and doctors' name, specialty, street, and city.

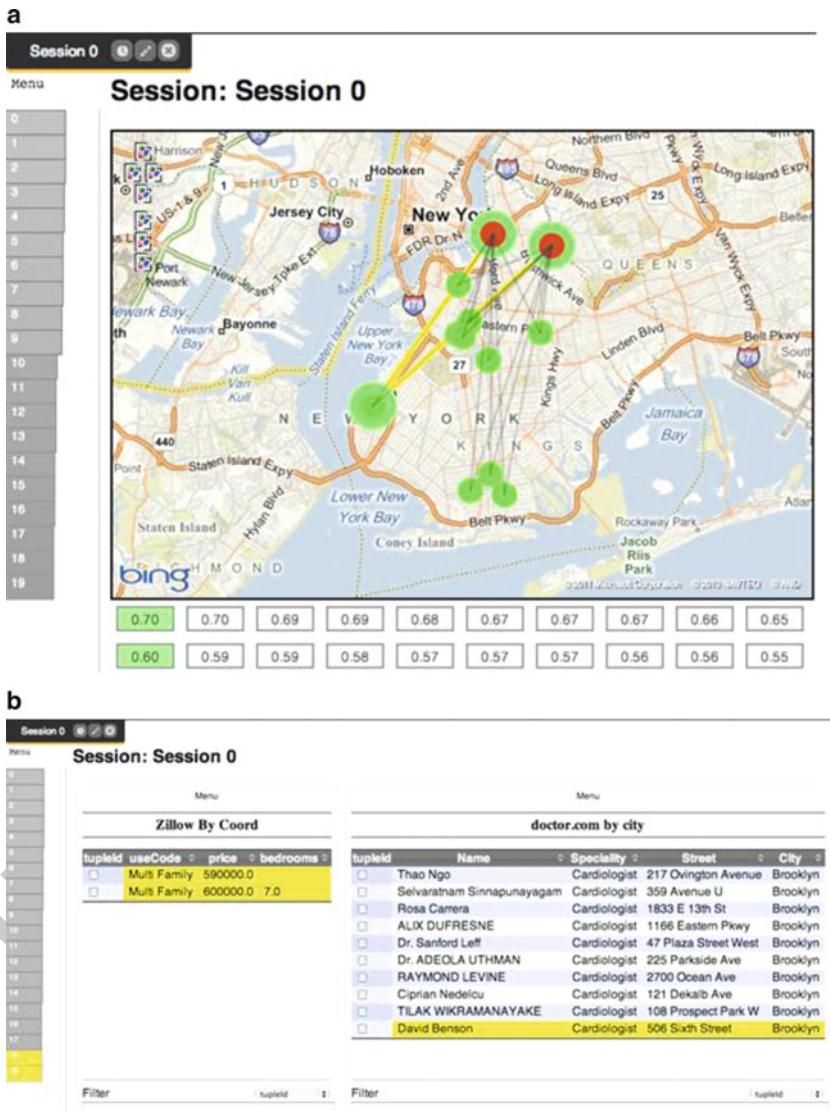


Fig. 3.11 Joint house offer/doctor exploration on the map (a) and atom view (b)

### 3.6 Deployment Architecture

318

The deployment of exploratory Web applications integrating data sources requires a 319 number of software components and quite sophisticated interactions between them; 320 in this section, we present the architecture currently under development within 321 the search computing system. The software modules in Fig. 3.12 are horizontally 322 divided into processing modules, repositories, and design tools, and vertically 323 organized as a two-tier, three-layer infrastructure, with the client tier dedicated 324 to user interaction and the server tier further divided into a control layer and 325 execution layer; the client–server separation occurs between processing layers and 326 repositories, and communications are performed using Web-enabled channels. Tools 327 address the various phases of interaction. 328

The *processing modules* include the *service invocation framework*, in charge of 329 invoking services that query the data sources. Such services typically have few 330 input parameters (which are used to complete parametric queries) and produce 331 results constituted by a “chunk” of tuples, possibly ranked, each equipped with a 332 tuple-id; thus, a service call maps given input parameters to a given chunk of tuples. 333 The framework includes built-in wrappers for the invocation of several Web-based 334 infrastructures (e.g., YQL, GBASE), query endpoint (e.g., SPARQL), and resources 335 (e.g., WSDL- and REST-based Web services). It also supports the invocation of 336 legacy and local data sources. The *execution engine* is a data- and control-driven 337 query engine specifically designed to handle multidomain queries [11]. It takes as 338 input a reference to a query plan and executes it, by driving the invocation of the 339 needed services. The *control layer* is the controller of the architecture; it is designed 340 to handle several system interactions (such as user session management and query 341 planning), and it embodies the *query analyzer* (parsing, planning, and optimizing 342 the queries) and the *query orchestrator* (acting as a proxy toward all the internal 343 components of the platform through a set of APIs). The *user interaction layer* is the 344 front end of the SeCo system. 345

The *repository* contains the set of components and data storages used by the 346 system to persist the artifacts required for its functioning. On the server side, the 347 *Service mart repository* contains the description of the search services consumed 348 by the system, represented at different levels of abstraction. The *query repository* 349 and *results repository* are used in order to persist query definitions and query 350 results which are heavily used, while the *user repository* and *application repository* 351 store, respectively, a description of users accessing an application and of the 352 configuration files (query configuration, result configuration, etc.) required by the 353 user interface for an application. On the client side, three persistent repositories 354 have been designed, respectively, the *query descriptions*, which are used by the 355 high-level query processor as reference definitions of the query models; the *service* 356 *mart descriptions* and the *application descriptions*, managed by the user interface on 357 the user’s browser to persistently cache application’s configuration files and service 358 descriptions. 359

AQ4

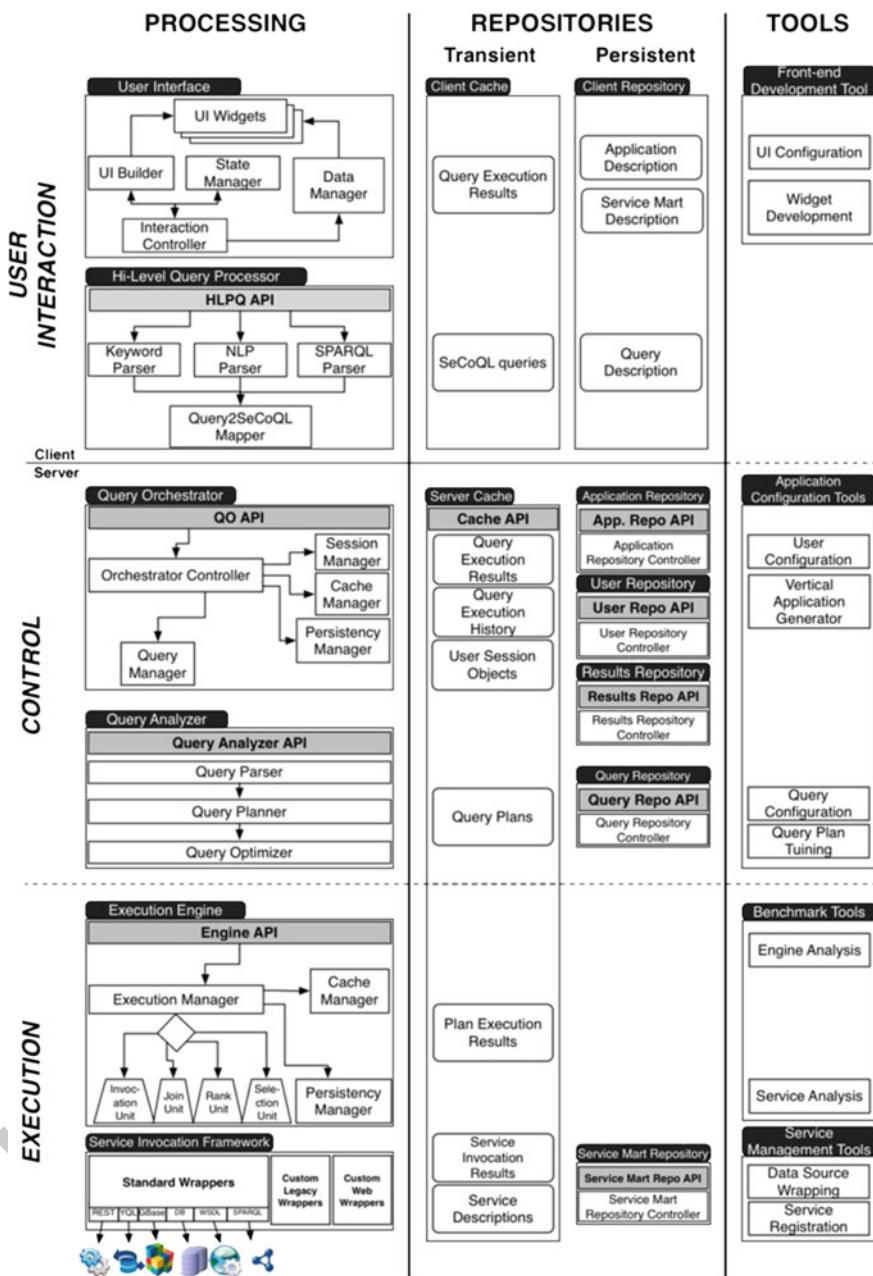


Fig. 3.12 The architecture of the search computing platform

To configure and design the set of objects and components involved in the SeCo architecture, a *tool suite* is currently under development as an online platform in which developers can login and, according to their role, access the right set of tools for building SeCo applications. The availability of the tools as online applications aims at increasing SeCo application design productivity, by reducing the time to deployment and avoiding the burden of downloading and installing software. The available tools include: the *service wrapping tool*, the *service registration tool* [12] (comprising the integration with the YAGO [23] general-purpose ontology for definition of terms), the *service analysis tool* for monitoring and evaluation of the service performance, the *execution analysis tool* for monitoring and fine-tuning the engine behavior, and the *application configuration tools* for allowing designers to define applications based on the composition of search services, along the principles discussed in Sect. 3.4. Several demos of these platforms have been presented at WWW [6], ACM-Sigmod [7], and ICWE [3].

### 3.7 Conclusions

374

In this chapter, we have presented our approach for designing vertical search applications that enable exploratory behavior upon structured Web data sources; the approach is general and consists of a set of guidelines and exploration patterns that can be exploited for designing a better user experience on the data exploration, based on the needs and peculiarities of each vertical domain.

Our approach is applicable to the infrastructures currently under development in the search computing project and has been validated on a few sample applications; we aim at a consolidation and wider experimentation in the future. Future work also includes a usability and user satisfaction evaluation, comparing the effectiveness of the different exploration strategies.

**Acknowledgements** This research is part of the search computing (SeCo) project, funded by ERC, under the 2008 Call for “IDEAS Advanced Grants” (<http://www.search-computing.org>). We wish to thank all the contributors to the project.

### References

388

1. Baeza-Yates, R., Broder, A., Maarek, Y.: The New Frontier of Web Search Technology: seven challenges. SeCO Workshop 2010, pp. 3–9. Springer LNCS (2010) 389  
390
2. Baeza-Yates, R.: Applications of web query mining. ECIR: European Conference on Information Retrieval, 2005, pp. 7–22. Springer LNCS 3408 (2005) 391  
392
3. Barbieri, D., Bozzon, A., Brambilla, M., Ceri, S., Pasini, C., Tettamanti, L., Vadacca, S., Volonterio, R., Zagorac, S.: Exploratory multi-domain search on web data sources with liquid queries. ICWE 2011 Conference, Demo session, June 2011, Paphos, Cyprus, 2011 393  
394
4. Bizer, C., Heath, T., Idehen, K., Berners-Lee, T.: Linked data on the web. WWW 2008, ACM, Beijing, China, 2008 395  
396  
397

- AQ5
5. Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P.: Liquid query: multi-domain exploratory search on the web. WWW '10, Raleigh, NC, ACM, New York, NY, USA, pp. 161–170, April 2010 398  
399  
400
  6. Bozzon, A., Brambilla, M., Ceri, S., Fraternali, P., Vadacca, S.: Exploratory search in multi-domain information spaces with Liquid Query. WWW 2011 Conference, Demo session, March 2011 401  
402  
403
  7. Bozzon, A., Brambilla, M., Ceri, S., Corcoglioniti, F., Fraternali, P., Vadacca, S.: Search computing: multi-domain search on ranked data. ACM-Sigmod 2011 Conference, Demo session, June 2011 404  
405  
406
  8. Braga, D., Campi, A., Ceri, S., Raffio, A.: Joining the results of heterogeneous search engines. Inf. Syst. **33**(7–8), 658–680 (2008) 407  
408
  9. Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Optimization of multi-domain queries on the web. VLDB '08, Auckland, NZ, 2008 409  
410
  10. Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Mashing up search services. IEEE Inter. Comput. **12**(5), 16–23 (2008) 411  
412
  11. Braga, D., Grossniklaus, M., Corcoglioniti, F., Vadacca, S.: Efficient computation of search computing queries. In: Ceri, S., Brambilla, M. (eds.) Search Computing Trends and Development. Springer LNCS vol. 6585, pp. 148–164 (2011) 413  
414  
415
  12. Brambilla, M., Tettamanti, L.: Tools supporting search computing application development. In: Ceri, S., Brambilla, M. (eds.) Search Computing Trends and Development. Springer LNCS vol. 6585, pp. 169–181 (2011) 416  
417  
418
  13. Broder, A.: A taxonomy of web search. SIGIR Forum **36**(2), 3, 10 (2002) 419
  14. Ceri, S., Brambilla, M. (eds.): Search Computing Trends and Development. Springer LNCS 6585 (2011) 420  
421
  15. Ceri, S., Brambilla, M. (eds.): Search Computing Challenges and Directions. Springer LNCS 5950, ISBN 978-3-642-12309-2 (2010) 422  
423
  16. Danescu-Niculescu-Mizil, C., Broder, A.Z., Gabrilovich, E., Josifovski, V., Pang, B.: Competing for users' attention: on the interplay between organic and sponsored search results. WWW 2010, Raleigh, NC, ACM, USA, pp. 291–300, April 2010 424  
425  
426
  17. Google: Fusion tables. <http://tables.googlelabs.com/> (2009) 427
  18. Ilyas, I., Beskales, G., Soliman, M.: A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. **40**(4) (2008) 428  
429
  19. Kules, B., Capra, R., Banta, M., Sierra, T.: What do exploratory searchers look at in a faceted search interface? JCDL, Joint Conference on Digital Libraries, pp. 313–322 (2009) 430  
431
  20. Marchionini, G.: Exploratory search: from finding to understanding. Commun. ACM **49**(4), 41–46 (2006) 432  
433
  21. Parameswaran, A., Das Sarma, A., Polyzotis, N., Widom, J., Garcia-Molina, H.: Human-assisted graph search: it's okay to ask questions. PVLDB **4**(5), 267–278 (2011) 434  
435
  22. Quarteroni, S.: Question answering, semantic search and data service querying. In: St-Dizier P. (ed.) 6th Workshop on Knowledge and Reasoning for Answering Questions (KRAQ'11), Chiang Mai, Thailand, November 2011 436  
437  
438
  23. Rajaraman, A.: Kosmix: high performance topic exploration using the deep web. P-VLDB **2**(2) (2009). Lyon, France 439  
440
  24. Suchanek, F., Bozzon, A., Della Valle, E., Campi, A.: Towards an ontological representation of services in search computing. In: Ceri, S., Brambilla, M. (eds.) Search Computing Trends and Development. Springer LNCS 6585, pp. 101–112 (2011) 441  
442  
443
  25. White, R.W., Roth, R.A.: Exploratory search. Beyond the Query, Response Paradigm. In: Marchionini, G. (ed.) Synthesis Lectures on Information Concepts, Retrieval, and Services Series, vol. 3. Morgan and Claypool, San Francisco (2009) 444  
445  
446
  26. Yahoo!. YQL. <http://developer.yahoo.com/yql/> (2009) 447

AUTHOR QUERIES

- AQ1. Kindly check the corresponding author.
- AQ2. Please check if ‘result’ can be changed to ‘resulting’ in the sentence “...behaviors and effectiveness of...”.
- AQ3. Please confirm the insertion of apostrophe in the sentence “...as we expect most services...”.
- AQ4. Please confirm the changes in the sentence “It takes in input a reference...”.
- AQ5. Kindly provide the accessed date for [17, 26].

UNCORRECTED PROOF

**Part II** 1  
**Search over the Web** 2

UNCORRECTED PROOF

## Chapter 4

# Path-Oriented Keyword Search Query over RDF

Roberto De Virgilio, Paolo Cappellari, Antonio Maccioni,  
and Riccardo Torlone

### 4.1 Introduction

We are witnessing a smooth evolution of the Web from a worldwide information space of linked documents to a global knowledge base, where resources are identified by means of uniform resource identifiers (URIs, essentially string identifiers) and are semantically described and correlated through resource description framework (RDF, a metadata data model) statements.

With the size and availability of data constantly increasing (currently around 7 billion RDF triples and 150 million RDF links), a fundamental problem lies in the difficulty users face to find and retrieve the information they are interested in. In general, to access semantic data, users need to know the organization of data and the syntax of a specific query language (e.g., SPARQL or variants thereof). Clearly, this represents an obstacle to information access for nonexpert users. For this reason, keyword search-based systems are increasingly capturing the attention of researchers. Recently, many approaches to keyword-based search over structured and semistructured data have been proposed [1, 11–14, 16, 18]. These approaches usually implement IR strategies [21, 22] on top of traditional database management systems with the goal of freeing the users from having to know data organization and query languages.

Typically, keyword-based search systems address the following key steps: identifying (substructures holding) data matching input keywords, linking identified data (substructures) into solutions (since data are usually scattered across multiple

AQ1

---

R.D. Virgilio (✉) · A. Maccioni · R. Torlone

Department of Informatics and Automation, University Roma Tre, Rome, Italy  
e-mail: [dvr@dia.uniroma3.it](mailto:dvr@dia.uniroma3.it); [maccioni@dia.uniroma3.it](mailto:maccioni@dia.uniroma3.it); [torlone@dia.uniroma3.it](mailto:torlone@dia.uniroma3.it)

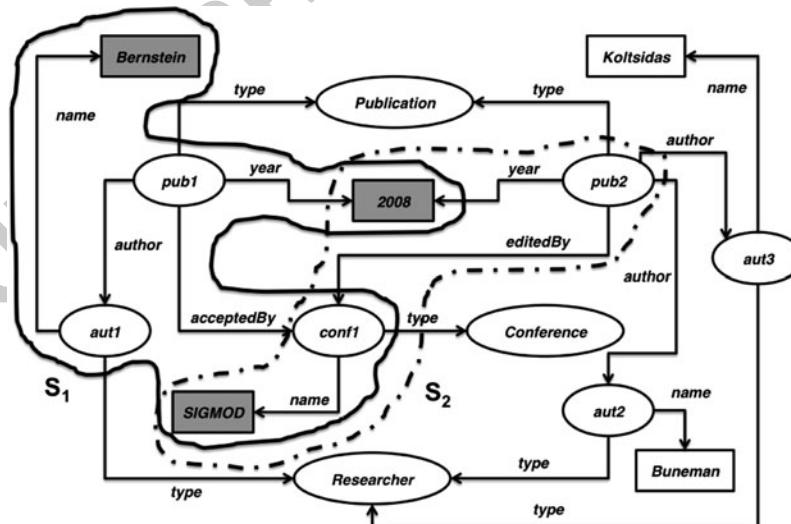
P. Cappellari  
Interoperable System Group, Dublin City, University, Dublin, Ireland  
e-mail: [pcappellari@computing.dcu.ie](mailto:pcappellari@computing.dcu.ie)

places, e.g., in different tables or XML elements), and ranking solutions according to a relevant criterion (i.e., a suitable scoring function). Eventually, only the top- $k$  solutions, i.e., those  $k$  solutions with highest score, are returned to the users as query answers.

For instance, let us consider the example in Fig. 4.1. It is inspired by the RDF version of the DBLP dataset (a database about scientific publications). Vertices in ovals represent an entity, such as  $aut1$  and  $aut2$ , or a concept, such as *Conference* and *Publication*. Vertices in rectangles are literal values, such as *Bernstein* and *Buneman*. Edges describe connections between vertices. For instance, the entity  $aut1$  is a *Researcher* whose name is *Bernstein*. In this example, given the query  $Q_1 = \{\text{Bernstein}, \text{SIGMOD}, 2008\}$ , two answers (i.e., the top 2) are  $S_1$  and  $S_2$ , in order, as depicted in Fig. 4.1. The vertices matching  $Q_1$  are emphasized in gray.

Given a query and a dataset, in order to obtain an exact answer, a process should first produce the candidate solutions, then filter and rank such solutions to return the final, exact, result. Referring to Fig. 4.1, the search task shall produce candidate solutions,  $S_1$  and  $S_2$  (and others, possibly), then the rank task orders and filters the these solutions to return  $S_1$  before  $S_2$  ( $S_1$  has higher score than  $S_2$ ). A limit of the current approaches is that they compute the best answers only in an approximate way (e.g., [1, 13]).

In this framework, search is a complex task that could produce many irrelevant candidates impacting on the quality of the final result. Recently, three measures have been proposed to evaluate this aspect [19]: *exhaustivity*, *specificity*, and *overlap*. Exhaustivity measures the relevance of a solution in terms of the number of keywords it contains. Specificity measures the precision of a solution in terms of the number of keywords it contains with respect to other irrelevant terms



**Fig. 4.1** An RDF graph from DBLP

occurring in the solution. Overlap measures the information content of a solution in terms of its intersection with other solutions. Clearly, the best ranking strategy balances exhaustivity and specificity while reducing overlap. In this respect, current approaches mainly focus on finding the most exhaustive solutions at the cost of a high level of overlapping. It turns out, however, that the quality of result is highly dependent on both the construction of candidate solutions and their ranking. For this reason, the problems of searching and ranking are strongly correlated.

Building on the above considerations, in this chapter, we present a novel keyword-based search technique over RDF graph-shaped data. We use a previous work [24] as starting point. That work focuses exclusively on computing qualitative results in an approximate order, ignoring efficiency and scalability issues. In this chapter, we merge a solution building strategy with a ranking method with the goal of generating the best  $k$  results in the first  $k$  produced answers, greatly reducing the complexity of the overall process. Referring to our example, in case of top-2 exploration, our proposal is able to produce in order  $S_1$  and  $S_2$  in the first two scans of the retrieval process.

Our search process proceeds as follows. The RDF graph is indexed off-line in order to efficiently retrieve the portions of interest for a query. The first task in evaluating a query consists of identifying the nodes (or edges) matching the query keywords. From the data graph point of view (i.e., RDF), we have sink nodes (i.e., classes and data values), intermediate and source nodes (i.e., URIs), and edges. As in [23], we can assume users enter keywords corresponding to attribute values (e.g., a name) rather than verbose URI; thus keywords refer principally to properties (i.e., edges) and literals (i.e., sinks). Under this assumption, we do not search URIs: this is not a limitation because nodes labeled by URIs are usually linked to literals which represent verbose descriptions of such URIs.

In our approach, we index all paths starting from a source and ending into a sink that we say represent the main flow of information in the graph. From a knowledge base paradigm point of view, these paths correspond to ontological assertions (i.e., A-Box) on an ontology vocabulary (i.e., T-Box). We cluster all assertions sharing the same portion of the vocabulary. Therefore, paths are clustered according to their *structure* (i.e., if they share edges) and, within each cluster, ranked according to their score. Solutions are generated by combining the most promising paths from each cluster. To combine such paths, we propose two different strategies: the first has a linear computational cost, whereas the second guarantees that at each step, the produced solution is the best possible (in the sense that the following solutions cannot have a higher rank). The first strategy enables the search to scale seamlessly with the size of the input, and the result shifts more to be exhaustive rather than specific. The second, inspired from the threshold algorithm proposed by Fagin et al. [5], guarantees the monotonicity of the output as we show that the first  $k$  solutions generated are indeed the top- $k$ : the result is optimally balanced between exhaustivity and specificity. In addition, we propose a variant of the second strategy that is both linear and monotonic. In this case, the result shifts more to be specific rather than exhaustive. Finally, all the strategies we propose avoid the generation of overlapping solutions.

To validate our approach, we have developed a system for keyword-based search 97 over RDF data that implement the techniques described in this chapter. Experiments 98 over widely used benchmarks have shown very good results with respect to other 99 approaches, in terms of both effectiveness and efficiency. 100

In sum, the contributions of our work are the following: 101

- A novel approach to keyword-based search over RDF data that combine an 102 exploration technique with a ranking mechanism to find best solutions first 103
- Two strategies for the construction of the query answers: one generating good 104 results quickly, the other generating best results first 105
- The development of a system for keyword-based search showing the improve- 106 ments that can be achieved compared to earlier approaches 107

The rest of the chapter is organized as follows. In Sect. 4.2, we introduce some 108 preliminary issues. In Sects. 4.3 and 4.4, we illustrate our strategies for keyword- 109 based search over RDF data. In Sect. 4.5, we discuss the related research, and in 110 Sect. 4.6, we present the experimental results. Finally, in Sect. 4.7, we draw our 111 conclusions and sketch future works. 112

## 4.2 Preliminary Issues

113

This section states the problem we address and introduces some preliminary notions 114 and terminology used during the presentation of our solution. 115

Informally, given a query as a list of keywords, we address the problem of 116 exploring a graph-shaped dataset to find subgraphs holding information relevant 117 to the query. 118

**Definition 4.1.** A labeled directed graph  $G$  is a five-element tuple  $G = \{V, E, 119 \Sigma_V, \Sigma_E, L_G\}$  where  $V$  is a set of vertices and  $E \subseteq V \times V$  is a set of ordered 120 pairs of vertices, called edges.  $\Sigma_V$  and  $\Sigma_E$  are the sets of vertices and edge labels, 121 respectively. The labeling function  $L_G$  defines the mappings  $V \rightarrow \Sigma_V$  and  $E \rightarrow \Sigma_E$ . 122

*Problem Statement.* Given a labeled directed graph  $G$  and a keyword search-based 123 query  $Q = \{z_1, z_2, \dots, z_n\}$ , where each  $z_i$  is a keyword, we aim at finding the top-k 124 ranked answers  $S_1, S_2, \dots, S_k$  to  $Q$ . 125

An answer  $S_i$  to a query  $Q$  is composed of connected subgraphs in  $G$  containing 126 at least one vertex matching a keyword  $z_i$ . Many proposals (e.g., [1, 13]) provide an 127 exact matching between keywords and labels of data elements. In our framework, 128 we adopt an IR approach to support an imprecise matching that involves syntactic 129 and semantic similarities. As a result, the user does not need to know the exact labels 130 of the data elements when doing keyword search. We say that a vertex  $v$  matches a 131 keyword when the label associated with a vertex (i.e.,  $L_G(v)$ ) contains the keyword 132 or a semantic expansion of the keyword. 133

**Definition 4.2 (Match).** Given a labeled directed graph  $G = \{V, E, \Sigma_V, \Sigma_E, L_G\}$  134 and a keyword search query  $Q$ , a vertex  $v \in V$  matches  $Q$  if there occurs one of the 135 following: 136

- There exists some keyword  $z_i \in Q$  whose value is equal to or contained in  $L_G(v)$ , 137 either lexically or on semantic query expansion. 138
- There exists some keyword  $z_i \in Q$  whose value is equal to or contained in the 139 label of an edge  $e$  (i.e.,  $L_G(e)$ ) directly connected with  $v$ , either lexically or 140 according to a semantic query expansion.<sup>1</sup> 141

Figure 4.1 depicts an example inspired by the RDF version of the DBLP dataset. 142 Vertices in ovals represent an entity, such as *aut1* and *aut2*, or a concept, such 143 as *Conference* and *Publication*. Vertices in rectangles are literal values, such as 144 *Bernstein* and *Buneman*. Edges describe connections between vertices. For instance, 145 the entity *aut1* is a *Researcher* whose name is *Bernstein*. 146

In this example, given the query  $Q_1 = \{\text{Bernstein, SIGMOD, 2008}\}$ , two 147 answers are  $S_1$  and  $S_2$  as depicted in Fig. 4.1. The vertices matching  $Q_1$  are 148 emphasized in gray. 149

The *sources* of a graph are those vertices  $v_{so}$  with no ingoing edges, that is, there 150 is no pair  $(v_i, v_{so}) \in E$  with  $v_i, v_{so} \in V$ . The *sinks* are the vertices  $v_{si}$  with no 151 outgoing edges, that is, there is not a pair  $(v_{si}, v_j) \in E$  with  $v_{si}, v_j \in V$ . Vertices in 152  $V$  that are not sources neither sinks are referred to as *intermediary vertices*. 153

*Data Structures.* In our approach, as explained in detail later, we avoid graph 154 traversal at query execution time by indexing information on the reachability of 155 vertices in terms of paths. In particular, we define a path as the sequence of vertices 156 and edges from a source to another vertex. 157

**Definition 4.3 (Full-Path).** Given a graph  $G = \{V, E, \Sigma_V, \Sigma_E, L_G\}$ , a full-path is 158 a sequence  $pt = l_{v_1} - l_{e_1} - l_{v_2} - l_{e_2} - \dots - l_{e_{n-1}} - l_{v_f}$  where  $v_i \in V$ ,  $e_i \in E$ , 159  $l_{v_i} = L_G(v_i)$ ,  $l_{e_i} = L_G(e_i)$  and  $v_1$  is a source. Each  $l_{v_i}$  and  $l_{e_i}$  are tokens in  $pt$ . 160

Each vertex  $v \in V$  is reachable by at least one full-path. Sources are special 161 vertices because they allow to reach any part of the (directed) graph. If a 162 source is not present, a fictitious one can be added. For instance, the graph 163 in Fig. 4.1 has two sources: *pub1* and *pub2*. An example of full-path is  $pt_k =$  164 *pub1-author-aut1-name-Bernstein*, having tokens *pub1*, *author*, 165 *aut1*, *name*, *Bernstein*. The length of a path is the number of vertices 166 occurring in the path, while the position of a vertex corresponds to the position into 167 the presentation order of all vertices. For instance,  $pt_k$  has length 3 and the vertex 168 *aut1* has position 2. 169

Answers to a query are built by assembling paths containing vertices matching 170 at least one keyword. Such paths represent our primary source of information, and 171 we refer to them as *sound paths*. 172

---

<sup>1</sup>We do not explore further the notion of semantic query expansion and refer here to standard IR techniques.

**Definition 4.4 (Sound Path).** Given a graph  $G$  and a query  $Q = \{z_1, \dots, z_n\}$ , a sound path is a full-path  $pt = l_{v_1} - l_{e_1} - l_{v_2} - l_{e_2} - \dots - l_{e_{n-1}} - l_{v_f}$  where at least  $v_f$  matches a keyword  $z_i \in Q$  (i.e., others  $v_i$  can match one or more keywords in case). We denote  $pt \models z_i$ .

For instance, with respect to the query  $Q_1$  and the graph depicted in Fig. 4.1, we have the following sound paths:

```

 $p_1 : \text{pub1-year-2008}$ 
 $p_2 : \text{pub1-author-aut1-name-Bernstein}$ 
 $p_3 : \text{pub1-acceptedBy-conf1-name-SIGMOD}$ 
 $p_4 : \text{pub2-year-2008}$ 
 $p_5 : \text{pub2-editedBy-conf1-name-SIGMOD}$ 

```

In a path, the sequence of the edge labels describes the corresponding structure. To some extent, such a structure describes a schema for the category of data (values on vertices) sharing the same connection types. While we cannot advocate the presence of a schema, we can say that such a sequence is a *template* for the path. Therefore, given a path  $pt$ , its template  $t_{pt}$  is the path itself where each vertex label in  $pt$  is replaced with the wildcard  $\#$ . In the example of reference, Fig. 4.1, the template  $t_{p_2}$  associated to  $p_2$  is  $\#-\text{author}-\#-\text{name}-\#$ . We say that  $p_2$  satisfies  $t_{p_2}$ , denoted with  $p_2 \approx t_{p_2}$ . Multiple paths can share the same template. Data belonging to paths sharing the same template can be considered as homogeneous. Therefore, we group paths into *clusters* with respect to the template they share: clusters partition the graph. For instance, we can group the sound paths from the graph of Fig. 4.1 as follows:

```

 $cl_1 : \#-\text{year}-\# \{p_1, p_4\}$ 
 $cl_2 : \#-\text{author}-\#-\text{name}-\# \{p_2\}$ 
 $cl_3 : \#-\text{acceptedBy}-\#-\text{name}-\# \{p_3\}$ 
 $cl_4 : \#-\text{editedBy}-\#-\text{name}-\# \{p_5\}$ 

```

The answer to a query is a combination of sound paths. When possible, each path belongs to a different cluster. Since we would correlate all the keywords, any pair of paths in a solution must intersect in a vertex at least. We define and denote the intersection between two paths as follows:

**Definition 4.5 (Intersection).** Given two paths  $pt_1$  and  $pt_2$ , there is an intersection between  $pt_1$  and  $pt_2$ , denoted by  $pt_1 \leftrightarrow pt_2$ , if  $\exists v_i \in pt_1$  and  $\exists v_j \in pt_2$  such that  $v_i = v_j$ .

For instance, the paths  $p_3$  and  $p_5$  have an intersection in `conf1` and `SIGMOD`. Finally, a solution  $S$  is a directed labeled graph built on a set of sound paths presenting pairwise intersections, as defined below.

**Definition 4.6 (Solution).** A solution  $S$  is a set of sound paths  $pt_1, pt_2, \dots, pt_n$  where  $\forall pt_a, pt_b \in S$ , there exists a sequence  $[pt_a, pt_{w_1}, \dots, pt_{w_m}, pt_b]$ , with

$m < n$ , such that  $pt_{w_i} \in S$ ,  $pt_a \leftrightarrow pt_{w_1}$ ,  $pt_b \leftrightarrow pt_{w_m}$ , and  $\forall i \in [1, m - 1] : pt_{w_i} \leftrightarrow pt_{w_{i+1}}$ .

Given a set of sound paths  $P = \{pt_1, pt_2, \dots, pt_n\}$ , we denote with  $S(\{pt_1, pt_2, \dots, pt_n\})$  (or  $S(P)$ ) to say that  $pt_1, pt_2, \dots, pt_n$  (or  $P$ ) represent a solution  $S$ . As for a sound path, if a solution  $S$  contains a vertex that matches a keyword  $z_i$ , then we denote  $S \models z_i$ .

On the graph illustrated in Fig. 4.1, given the query  $Q_1$ , the answers are represented by the solutions  $S_1(\{p_1, p_2, p_3\})$  and  $S_2(\{p_4, p_5\})$ .

**Scoring Function.** A fundamental feature of search systems is the ability to return the solutions to a query in order of relevance. To this end, we use a scoring function to rank the candidate paths. Solutions are ranked according to a scoring function as well. Note that, as will be clarified during the discussion, our approach is scoring function independent. We here define a scoring function for completeness of presentation and evaluation, as this function is later needed to run our experiments. In literature, several metrics have been proposed [10, 21–23]. We define a single scoring function to assess the relevance of a subgraph with respect to the query of reference. In fact, while a solution is a proper subgraph, a sound path can be seen as the simplest possible subgraph. We assess a subgraph on two key factors: its topology and the relevance of the information carried from its vertices and edges. The topology is evaluated in terms of the length of its paths. This is a basic metric for ranking answer widely used in recent approaches to keyword search (e.g., [11]). The relevance of the information carried from its vertices is evaluated through an implementation of TF/IDF (term frequency and inverse document frequency).

**Definition 4.7.** Given a query  $Q = \{z_1, \dots, z_n\}$ , a graph  $G$ , and a subgraph  $sg$  in  $G$ , the score of  $sg$  with respect to  $Q$  is

$$score(sg, Q) = \frac{\alpha(sg)}{\omega_{str}(sg)} \cdot \sum_{z \in Q} (\rho(z) \cdot \omega_{ct}(sg, z))$$

where:

- $\alpha(sg)$  is the relative relevance of  $sg$  within  $G$
- $\rho(z)$  is the weight associated to each keyword  $z$  with respect to the query  $Q$
- $\omega_{ct}(sg, z)$  is the *content* weight of  $z$  considering  $sg$
- $\omega_{str}(sg)$  is the *structural* weight of  $sg$

The relevance of  $sg$  is measured as follows:

$$\alpha(sg) = \left( \prod_{v_i \in sg} \left( \frac{\deg^-(v_i) + 1}{\deg^+(v_i) + 1} \right) \right)^{\frac{1}{|V(sg)|}}$$

Inspired by link analysis algorithms (e.g., PageRank [2]), we consider the relevance of each vertex  $v_i \in V$  as the ratio between the in-degree ( $\deg^-$ ) and the

out-degree ( $\deg^+$ ) of  $v_i$ . The relevance of  $sg$  is the geometric mean<sup>2</sup> of the relevance of each vertex  $v_i \in sg$ . In  $\alpha(sg)$ , we indicate the number of vertices in  $sg$  as  $|V(sg)|$ .

The scoring supports the specification of weights on the input keywords, i.e.,  $\rho(z)$ . For instance, we can imagine that the order of the keywords in the query represents the relevance of each keyword. While our system supports the specification of such relevant information, in the remaining of the discussion, we assume the input keywords have all the same relevance, meaning that for a hypothetical user, all the keywords have the same relevance (i.e.,  $\rho(z) = 1, \forall z$ ).

In the scoring function,  $\omega_{ct}$  aims to capture the *commonness* of a keyword  $z$  in  $sg$ . The commonness measures the relative number of graph elements which it actually represents. The higher the commonness, the lower its contribution should be. This function has been inspired by the pivoted normalization weighting method [21, 22], one of the most used metric in IR. Formally, it is defined as follows:

$$\omega_{ct}(sg, z) = \begin{cases} \left(1 + \frac{tf}{df_{sg}(z)}\right) \cdot \left(1 + \frac{|\mathcal{PT}|}{DF(z)+1}\right), & \text{if } sg \models z \\ 0, & \text{otherwise} \end{cases}$$

where:

- $tf$  is the number of times a keyword  $z$  matches a vertex or an edge in  $sg$ , and  $df_{sg}(z)$  is the number of paths of  $sg$  where  $z$  has a match (i.e., if  $sg$  is a path then  $df_{sg}(z) = 1$ )
- $|\mathcal{PT}|$  and  $DF(z)$  are the number of all sound paths  $\mathcal{PT}$  matching  $Q$  and the number of paths in  $\mathcal{PT}$  having a match with  $z$ , respectively

Factor  $\omega_{str}$  exploits the structural features of  $sg$ , evaluating the *proximity* (distance) of the keywords in  $sg$ . The higher the structural weight, the lower the proximity and its contribution should be. In other terms, a shorter path between two keyword elements should be preferred. We define

$$\omega_{str}(sg) = |V(sg)|$$

where  $|V(sg)|$  is the number of vertices in  $sg$ . The structural weight normalizes the content weight. In this way, we have a score balanced between content and structural features, i.e., between specificity and exhaustivity.

For instance, let us consider the score of  $p_1$  in our running example with respect to the query  $Q_1$ . The vertices pub1 and 2008 present the relative relevances (i.e.,  $\frac{\deg^-(v_i)+1}{\deg^+(v_i)+1}$ ) 0.2 and 3, respectively. The path  $p_1$  has  $\alpha(p_1) = \sqrt{0.2 \cdot 3} = 0.77$  and  $\omega_{str}(p_1) = 2$ , having two vertices. Since we assume all keywords in  $Q_1$  have the same relevance,  $\rho(z_i) = 1, \forall z_i \in Q_1$ . Finally, we have to calculate  $\omega_{ct}$ . Given  $|\mathcal{PT}| = 5$  and  $DF(2008) = 2$ , we have

---

<sup>2</sup>The geometric mean of a dataset  $\{a_1, a_2, \dots, a_n\}$  is given by  $\sqrt[n]{a_1 \cdot a_2 \cdots a_n}$ .

$\omega_{ct}(p_1, \text{Bernstein}) = \omega_{ct}(pt_1, \text{SIGMOD}) = 0$  and  $\omega_{ct}(pt_1, 2008) = (1+1) \cdot (1 + \frac{5}{1+2}) = 5.33$ . At the end, we have  $score(p_1, Q_1) = 2.05$ .

272  
273

## 4.3 Linear Strategy for Solution Building

274

The approach is composed of two main phases: an *off-line indexing*, where the graph  $G$  is indexed in order to have immediate access to the information of interest, and the *keyword processing* (on the fly), where the query evaluation takes place. The first task is efficiently performed by implementing an index structure, as we will describe in Sect. 4.6. In the second task, given the query  $Q$  our framework retrieves all sound paths  $\mathcal{PT}$  by using the index and explores the best solutions from  $\mathcal{PT}$  through a strategy that guarantees a linear time complexity with respect to the size of the input  $\mathcal{PT}$ , that is,  $I = |\mathcal{PT}|$ . Before computing solutions, we organize the paths of  $\mathcal{PT}$  into clusters.

283

### 4.3.1 Clustering

284

Given the list  $\mathcal{PT}$ , we group the paths into clusters according to their template, and we return the set  $\mathcal{CL}$  of all the clusters. Therefore,  $\mathcal{CL}$  is computed as shown in Algorithm 1.

285  
286  
287

The set of clusters is implemented as a map where the key is a template and the value is a cluster with such template associated. Each cluster is implemented as a priority queue of paths, where the priority is based on the score (descending) associated to each path. For each  $pt \in \mathcal{PT}$  (line [2]), we search the cluster  $c1$  having associated the same template  $t_{cl}$  corresponding to the schema of  $pt$  (line [3]). If there does not exist such cluster, we create a new one (line [4]). At the end, we will insert  $pt$  in  $c1$  (line [5]) and we update the map  $\mathcal{CL}$  with the new  $c1$  with the template of  $pt$  ( $t_{pt}$ ) as key (line [6]). Referring to the example of Fig. 4.1, we have the set of clusters  $\mathcal{CL} = \{c1, c2, c3, c4\}$ . In particular in  $c1$ , we have  $p_1$  more relevant than  $p_2$  (i.e.,  $score(p_1, Q_1) > score(p_2, Q_1)$ ). It is

288  
289  
290  
291  
292  
293  
294  
295  
296  
297
**Algorithm 1:** Clustering of paths

**Input :** A list of sound paths  $\mathcal{PT}$ .  
**Output:** The map  $\mathcal{CL}$  of the clusters.

```

 $\mathcal{CL} \leftarrow \emptyset;$ 
foreach  $pt \in \mathcal{PT}$  do
    if  $\exists(t_{cl}, cl) \in \mathcal{CL} : pt \approx t_{cl}$  then
         $cl \leftarrow \emptyset;$ 
         $cl.Enqueue(pt);$ 
         $\mathcal{CL}.put(t_{pt}, cl);$ 
    return  $\mathcal{CL};$ 

```

**Algorithm 2:** Building solutions in linear time

---

**Input :** The map  $\mathcal{CL}$ , a number  $k$ .  
**Output:** A list  $\mathcal{S}$  of  $k$  solutions.

```

 $S \leftarrow \emptyset;$ 
while  $|\mathcal{S}| < k$  and  $\mathcal{CL}$  is not empty do
| First  $\leftarrow \emptyset$ ;
| CC  $\leftarrow \emptyset$ ;
| foreach  $cl \in \mathcal{CL}$  do
| | First  $\leftarrow$  First  $\cup$   $cl.\text{DequeueTop}()$  ;
| | CC  $\leftarrow \text{FindCC}(First)$  ;
| | ccSols  $\leftarrow \emptyset$ ;
| | foreach  $cc \in CC$  do
| | | sol  $\leftarrow \text{newSolution}(cc)$ ;
| | | ccSols.Enqueue(sol);
| | S.InsertAll(ccSols.DequeueTop(k - |S|));
| return  $\mathcal{S}$ ;
```

---

straightforward to demonstrate that the time complexity of the clustering is  $O(I)$ : 298  
we have to execute  $I$  insertions into  $\mathcal{CL}$  at most. 299

**4.3.2 Linear Building**

300

Given the set of clusters  $\mathcal{CL}$ , the building of solutions is performed by generating 301  
the connected components  $CC$  from the most promising paths in  $\mathcal{CL}$ . Algorithm 2 302  
illustrates the main steps of the process. 303

The algorithm iterates  $k$  times at most to produce the best  $k$  solutions (i.e., a 304  
list  $\mathcal{S}$ ). At the beginning of each iteration, we initialize the set  $First$  containing 305  
the most relevant paths from each cluster, that is, the paths with the highest score 306  
(lines [3–6]). The task is supported by the function  $\text{DequeueTop}$  that extracts from 307  
the cluster  $cl$  the top score path (i.e., multiple paths if they have the same score). 308  
Then we compute  $CC$  from  $First$  (line [7]), supported by the function  $\text{FindCC}$ , 309  
and for each component, we generate a new solution (lines [8–11]) by using the 310  
function  $\text{newSolution}$ . The resulting set of solutions  $ccSols$  is implemented 311  
as a priority queue with respect to the score of a solution. From  $ccSols$ , we insert 312  
the top  $n$  elements into  $\mathcal{S}$ , where  $n$  results from the difference between  $k$  and the 313  
actual size of  $\mathcal{S}$  (i.e.,  $|\mathcal{S}|$ ). In this case, we use a variant of  $\text{DequeueTop}$ , giving  $n$  314  
as input to extract the top- $n$  elements, and  $\text{InsertAll}$  to insert the solutions into 315  
 $\mathcal{S}$  (line [12]). The execution concludes when we produced  $k$  solutions (i.e.,  $|\mathcal{S}| < k$ ) 316  
or  $\mathcal{CL}$  became empty. 317

Algorithm 3 describes the function  $\text{FindCC}$ . It iterates on the set  $First$ : a path 318  
 $pt$  is extracted and a supporting recursive function  $\text{FindCC\_AUX}$  computes the 319  
connected component between  $pt$  and the updated  $First$  (lines [3–4]). As shown 320  
in Algorithm 4,  $\text{FindCC\_AUX}$  searches all paths  $pt_i \in First$  such that  $pt_i$  and 321

**Algorithm 3:** Find  $CC$  from a set of paths

---

**Input :** A set First of paths.  
**Output:** A set CC of connected components in First.

```

CC ← ∅;
while First is not empty do
    First ← First \ {pt};
    CC ∪ FindCC_AUX(pt, First )
return CC;
```

---

**Algorithm 4:** FindCC\_AUX: a recursive function supporting FindCC

---

**Input :** A path pt, a set of paths First.  
**Output:** The connected component built from pt.

```

PQ ← ∅;
PQ ← PQ ∪ {pt};
foreach  $pt_i \in First: pt_i \leftrightarrow pt$  do
    First ← First -  $pt_i$ ;
    PQ ← PQ ∪ FindCC_AUX( $pt_i$ , First );
return PQ ;
```

---

$pt$  present an intersection (i.e.,  $pt_i \leftrightarrow pt$ ). Until there are paths  $pt_i$  having an 322  
 intersection with  $pt$ , we extract  $pt_i$  from First, insert them into the final result 323  
 $PQ$ , and we call the recursion (lines [4–5]). 324

Algorithm 2 produces the best-k solutions in linear time with respect to the 325  
 number  $I$  of sound paths matching the input query  $Q$ : it is in  $O(k \times I) \in O(I)$ . 326  
 In the worst case, the algorithm iterates  $k$  times. The execution in lines [4–5] is 327  
 $O(|(CL|)) \in O(I)$ . Then we have to evaluate the execution of FindCC: since 328  
 such function calls FindCC\_AUX for each connected component (i.e., at most for 329  
 each path within First), the complexity of the task is  $O(I)$ . Finally, both the 330  
 executions in lines [9–11] and line [12] are in  $O(I)$  (i.e., at most we have to make 331  
 $I$  insertions). Therefore, the entire sequence of operations in the Algorithm 2 is in 332  
 $O(k \times I) \in O(I)$ . 333

Referring again to the example of Fig. 4.1, in the first iteration of the algorithm, 334  
 the most promising paths from each cluster are First =  $\{p_1, p_2, p_3, p_5\}$ . FindCC 335  
 extracts from First the connected component  $cc_1$  containing all the paths 336  
 (i.e., they have pairwise intersections). Therefore,  $cc_1$  will be returned as the first 337  
 solution  $S_1(\{p_1, p_2, p_3, p_5\})$ . At the second iteration, First =  $\{p_4\}$ : the second 338  
 solution will be  $S_2(\{p_4\})$ . Since  $CL$  is empty, the algorithm ends at the second 339  
 iteration. In this example, we do not have overlap, and in particular, we obtain a 340  
 solution  $S_1$  exhaustive but not optimally specific (i.e.,  $p_5$  is excess). This suggests 341  
 that such strategy tends to shift more to be exhaustive rather than specific. Moreover, 342  
 it cannot assure the monotonicity of the solutions: it may happen the generation of 343  
 a sequence of two solutions  $S_i$  and  $S_{i+1}$  where  $score(S_{i+1}, Q) > score(S_i, Q)$ . 344  
 Let us suppose to generate a first solution  $S_1$  containing more than one path and 345

in particular containing the path  $pt'_1$  with the highest score. Then at the second iteration, we have a path  $pt'_2$  with the second most high score that does not present an intersection with any other extracted paths. Therefore, we will have a second solution  $S_2(\{pt'_2\})$ . In this case, if the scores of  $pt'_1$  and  $pt'_2$  are comparable, then it may happen that  $score(S_2, Q) = score(pt'_2, Q) > score(S_1, Q)$  due to the more complex structural weight of  $S_1$ . Of course the scores of the two solutions could be comparable, but the monotonicity is violated.

Since monotonicity is our most important target, in the next section we provide a variant of our algorithm to explore top-k solution that guarantees such property. In this way, our framework will be able to allow the exploration of top-k solution in both linear and monotonic ways. A strong point of this algorithm is the independence from ad hoc scoring functions: it requires very general properties on the way to rank solutions.

## 4.4 Monotonic Strategy for Solution Building

359

Monotonicity of the building is a relevant challenge in keyword search systems. This means to return the optimal solution at each generation step instead of waiting the processing of blocks of candidate solutions and then selecting the optima. In this section, we provide a second strategy allowing such capability. An important point of this algorithm is a joint use of the scoring function at the generation time of a solution. It introduces a method to establish if the solution we are generating is the optimum. Moreover, an optimization allows to guarantee the execution in linear time also.

367

### 4.4.1 Scoring Properties

368

For our purposes, we make some assumptions on the scoring function. Such assumptions represent general properties to guarantee the monotonicity.

370

**Property 1** Given a query  $Q$  and a path  $pt$ , we have

371

$$score(pt, Q) = score(S(\{pt\}), Q)$$

It means that every path has to be evaluated as well as the solution containing exactly that path. The property allows us to avoid the case in which  $score(pt_1, Q) > score(pt_2, Q)$  and  $score(S(\{pt_1\}), Q) < score(S(\{pt_2\}), Q)$ .

374

Moreover, let us consider a set of paths  $P = \{pt_1, pt_2, \dots, pt_n\}$  and  $P^*$  the power set of  $P$ . We provide a second property as follows:

376

**Property 2** Given a query  $Q$ , a set of path  $P$ , and the path  $pt_\beta \in P$ , such that  $\forall pt_j \in P : score(pt_\beta, Q) \geq score(pt_j, Q)$ , with  $pt_\beta \neq pt_j$ , we have

378

$$\text{score}(S(P_i), Q) \leq \text{score}(S(\{pt_\beta\}), Q) \quad \forall P_i \subseteq P^*$$

In other words, given the set  $P$  containing the candidate paths to be involved into a solution, the scores of all the possible resulting solutions generated from  $P$  (i.e.,  $P^*$ ) are bounded by the score of the most relevant path  $pt_\beta$  of  $P$ . Such property is supported by the threshold algorithm (TA) [5]. TA computes the top-k objects with the highest scores from a list of candidates: the score of each object is computed by an aggregation function, assumed monotone, on top of the individual scores obtained for each of the object's attributes. Iteratively, the process includes objects into the result until the score of the k-ranked object is higher than the upper bound score of all remaining objects. Compared to TA, here the object is a subgraph resulting by the best composition of sound paths that are the attributes. However, we do not use an aggregation function, nor do we assume the aggregation to be monotone. TA introduces a mechanism to optimize the number of steps  $n$  to compute the best  $k$  objects (where it could be  $n > k$ ), while our framework tries to produce  $k$  optimal solutions in  $k$  steps.

Although the above properties refer to the data structures used in our framework, they are very general. For instance, we can demonstrate that the two properties are fulfilled by the most common IR-based approaches. To this aim, we refer to the pivoted normalization weighting method (SIM) [22], which inspired most of the IR scoring functions. Let us consider a collection of documents  $D = \{D_1, \dots, D_m\}$  to evaluate with SIM against a query  $Q'$ . PROPERTY 1 is trivial: the score of any document  $D_i \in D$  is the same as a document included into an answer to  $Q'$  as a complete answer. Now let  $D^* = \{D'_1, D'_2, \dots, D'_n\}$  the power set of  $D$  and  $D_\beta$  the most relevant document in  $D$  with respect to  $Q'$ . Each document can be considered as a set of terms possibly matching a keyword of  $Q'$ , while each  $D'_i \in D^*$  results from the concatenation of two or more documents in  $D$ . We remind that SIM weights a keyword  $z$  of  $Q'$  into a document  $D_i$  as  $\text{weight}(z, D_i) = \frac{\text{ntf}(D_i)}{\text{ndl}(D_i)} \cdot \text{idf}$ . In such weight,  $\text{ntf}$  evaluates the number of occurrences of  $z$  into  $D_i$ ,  $\text{ndl}$  measures the length of the document  $D_i$  (i.e., number of terms), and  $\text{idf}$  quantifies the document frequency, that is, the number of documents in which  $z$  occurs. All of these measures are normalized:  $\text{ntf}$  by applying the  $\log$  function twice,  $\text{ndl}$  by the ratio with the average document length in the collection  $D$ , and  $\text{idf}$  by applying the  $\log$  function on the ratio between the total number of documents (i.e.,  $|D|$ ) and the document frequency. To verify PROPERTY 2, we have to show that  $\text{weight}(z, D_\beta) \geq \text{weight}(z, D'_i)$ ,  $\forall D'_i \in D^*$ . Since  $\text{idf}$  is the same in both the terms, we can consider just  $\text{ntf}$  and  $\text{ndl}$ . If  $D_\beta$  has the highest score, then  $\text{weight}(z, D_\beta)$  presents the best percentage of occurrences of  $z$  on top of the number of terms in the document. So given  $D'_i \in D^*$ , if  $\text{ntf}(D_\beta)$  and  $\text{ntf}(D'_i)$  are often comparable (i.e., due by applying the  $\log$  function twice),  $\text{ndl}(D_\beta)$  is less than  $\text{ndl}(D'_i)$  (i.e., relevantly through the normalization): at the end, we have  $\frac{\text{ntf}(D_\beta)}{\text{ndl}(D_\beta)} \geq \frac{\text{ntf}(D'_i)}{\text{ndl}(D'_i)}$ , that is,  $\text{weight}(z, D_\beta) \geq \text{weight}(z, D'_i)$ . Therefore, SIM satisfies the PROPERTY CH05:PROP02. Similarly, we can demonstrate that also our scoring function satisfies the two properties.

#### 4.4.2 Checking the Monotonicity

421

To check the monotonicity, we introduce a *threshold*  $\tau$ . Such threshold is functional 422  
to define the so-called  $\tau$ -*test*. As in Algorithm 2, we start computing the connected 423  
components from a set containing the most important paths of each cluster. Then, 424  
differently from the previous strategy, we do not verify only the intersection between 425  
paths but in particular if and how much it is opportune to insert a path into a partially 426  
generated solution. This task is supported by  $\tau$ -*test*. 427

First of all, we have to introduce some notations as follows: 428

- $cc$  represents a connected component containing candidate paths to be inserted 429  
into a solution. 430
- $s_{cc} \subset cc$  is the set of paths just inserted into the optimal solution (i.e., the set of 431  
paths that satisfied  $\tau$ -*test* so far). 432
- $pt_s$  is the path, not yet extracted from the set of clusters  $\mathcal{CL}$ , with the highest 433  
score in  $\mathcal{CL}$ . 434
- $pt_x$  is a path in  $\{cc \setminus s_{cc}\}$  (i.e., a candidate path to insert into  $s_{cc}$ ). 435
- $pt_y$  is the path in  $\{cc \setminus s_{cc}\}$  with the highest score. 436

Then we define the threshold  $\tau$  as 437

$$\tau = \max\{score(pt_s, Q), score(pt_y, Q)\}$$

$\tau$  can be considered as the upper bound score for the potential solutions to generate 438  
in the next iterations of the algorithm. 439

Now, we provide the following theorem: 440

**Theorem 4.1.** *Given a query  $Q$ , a scoring function satisfying the properties 441  
PROPERTY 1 and PROPERTY 2, a connected component  $cc$ , a subset  $s_{cc} \subset cc$  442  
representing an optimal solution, and a path  $pt_x \in \{cc \setminus s_{cc}\}$ ,  $S(s_{cc} \cup \{pt_x\})$  443  
is still optimum iff 444*

$$score(S(s_{cc} \cup \{pt_x\}), Q) \geq \tau$$

The condition provided by Theorem 4.1 is the  $\tau$ -*test*. 445

*Proof.* [Necessary Condition]. Let us assume that  $S(s_{cc} \cup \{pt_x\})$  is an optimal 446  
solution. We have to verify if the score of such solution is still greater than  $\tau$ . 447  
Reminding to the definition of  $\tau$ , we can have two cases: 448

- $\tau = score(pt_s, Q) > score(pt_y, Q)$ . 449

In this case,  $score(pt_s, Q)$  represents the upper bound for the scoring of the 450  
possible solutions to generate in the next steps. Recalling PROPERTY 1, we 451  
have  $score(pt_s, Q) = score(S(\{pt_s\}), Q)$ . Referring to PROPERTY 2, the 452  
possible solutions to generate will present a score less than  $score(S(\{pt_s\}), Q)$ : 453  
 $S(s_{cc} \cup \{pt_x\})$  is optimum. Therefore,  $score(S(s_{cc} \cup \{pt_x\})) \geq \tau$ . 454

$$\bullet \quad \tau = score(pt_y, Q) > score(pt_s, Q).$$

In a similar way,  $score(S(s_{cc} \cup \{pt_x\}), Q) \geq \tau$ .

455

456

**[Sufficient Condition].** Let us consider  $score(S(s_{cc} \cup \{pt_x\}), Q) \geq \tau$ . We have to verify if  $S(s_{cc} \cup \{pt_x\})$  is an optimal solution. From the assumption,  $score(S(s_{cc} \cup \{pt_x\}), Q)$  is greater than both  $score(pt_s, Q)$  and  $score(pt_y, Q)$ . Recalling again the properties of the scoring function, the possible solutions to generate will present a score less than both  $score(S(pt_s), Q)$  and  $score(S(\{pt_y\}), Q)$ . Therefore,  $S(s_{cc} \cup \{pt_x\})$  is an optimal solution.  $\square$

457

458

459

460

461

462

#### 4.4.3 Monotonic Building

463

As in Sect. 4.3, we start from a set of clusters  $\mathcal{CL}$ , grouping all informative paths matching the input query  $Q$  with respect to the shared template. To generate the top-k solutions guaranteeing the monotonicity, in the building algorithm of Sect. 4.3, we introduce an exploration procedure employing the  $\tau$ -test to check if the solution we are generating is (still) optimum. Algorithm 5 illustrates the new building process. As in Algorithm 2, the process starts computing the connected components from the top paths of each cluster (lines [2–6]). Once extracted such paths, if the set of clusters  $\mathcal{CL}$  is not empty, we can activate the  $\tau$ -test (lines [7–10]); otherwise, since we have all possible paths in the connected components, we proceed as in Algorithm 2 (lines [12–15]).

473

---

**Algorithm 5:** Monotonic building of top- $k$  solutions

---

**Input :** A list  $\mathcal{CL}$  of clusters, a number  $k$ .

**Output:** A list  $S$  of  $k$  solutions.

```

while  $|S| < k$  do
    First  $\leftarrow \emptyset$ ;
    CC  $\leftarrow \emptyset$ ;
    foreach cl  $\in \mathcal{CL}$  do
        First  $\leftarrow$  First  $\cup$  cl.DequeueTop();
        CC  $\leftarrow$  FindCC(First);
    if  $\mathcal{CL}$  is not empty then
         $pt_s \leftarrow$  getTopPath( $\mathcal{CL}$ );
        BSols  $\leftarrow$  MonotonicityExploration(CC,  $\mathcal{CL}$ ,  $pt_s$ );
        S.InsertAll(BSols);
    else
        foreach cc  $\in$  CC do
            sol  $\leftarrow$  newSolution(cc);
            ccSols.Enqueue(sol);
        S.InsertAll(ccSols.DequeueTop(k - |S|));
    return S;

```

---

**Algorithm 6:** Monotonicity exploration

---

**Input :** A set CC of connected components, a list  $\mathcal{CL}$  of clusters, a path  $pt_s$ .  
**Output:** A list of solutions BSols.

```

CCOpt ← ∅;
foreach cc ∈ CC do
    CCOpt.Enqueue( MonotonicityAnalysis(cc, ∅, pts) );
BSols.InsertAll( CCOpt.DequeueTop() );
InsertPathsInClusters( CCOpt, CL );
return BSols ;

```

---

In the  $\tau$ -test, we calculate the top path  $pt_s$  from  $\mathcal{CL}$  and we activate the 474 exploration to find the optimal solutions. The process is supported by the function 475 getTopPath to extract  $pt_s$  from  $\mathcal{CL}$  and by the function Monotonicity 476 Exploration to find the best solutions to insert into  $\mathcal{S}$ . The function 477 MonotonicityExploration is described in Algorithm 6. It takes as input 478 the set of connected components CC, the set of clusters  $\mathcal{CL}$ , and the path  $pt_s$ . This 479 function has to extract from each connected component cc the optimal subset of 480 paths (i.e., representing the optimal solution). Such subsets are collected into a 481 priority queue CCOpt (lines [2–3]). Therefore, the top elements of CCOpt will 482 correspond to the optimal solutions: we implement a list BSols of such solutions 483 (line [4]). At the end, all excluded paths in CCOpt will be reinserted into  $\mathcal{CL}$  by 484 using the function InsertPathsInClusters (line [5]). 485

The queue CCOpt is computed by the analysis of monotonicity through the 486  $\tau$ -test. The analysis is performed by the recursive function MonotonicityAnalysis, 487 as shown in Algorithm 7. It takes as input the connected component cc, the current 488 optimal solution OptS, and the top path  $pt_s$  contained in  $\mathcal{CL}$ . The idea is to 489 generate OptSols, that is, the set of all solutions (candidate to be optimal) by 490 combining OptS with the paths of cc. If cc is empty, we return OptS as it is 491 (i.e., we cannot explore more). Otherwise, we analyze all paths  $pt_x \in cc$  that 492 present an intersection with a path  $pt_i$  of OptS. If there is not any intersection, 493 then OptS is the final optimal solution (lines [6–7]). Otherwise, for each  $pt_x$ , we 494 calculate  $\tau$  (line [9]), through the function getTau, and we execute the  $\tau$ -test 495 on each new solution tryOptS, that is,  $OptS \cup \{pt_x\}$ . If tryOptS satisfies the 496  $\tau$ -test, then it represents the new optimal solution: we insert it into OptSols and 497 we invoke the recursion on tryOptS (lines [9–12]). Otherwise, we keep OptS 498 as optimal solution and skip  $pt_x$  (line [14]). At the end, we take the best OptS 499 from OptSols by using TakeMaximal (line [15]) described in Algorithm 8. 500 Such function selects the solution from OptSols that contains more sound 501 paths firstly (line [4]) and then presents the highest score (line [6]). Let us 502 consider our running example. As for the linear strategy, in the first iteration of 503 the algorithm, we start from First =  $\{p_1, p_2, p_3, p_5\}$ . The paths of First have 504 scores 2.05, 1.63, 1.6, and 1.49, respectively. Now the exploration will consider 505 all possible combinations of such paths to find the optimal solution(s). Therefore, 506 at the beginning, we have OptS =  $\{p_1\}$ , since  $p_1$  has the highest score, and  $pt_s$  507

**Algorithm 7:** Monotonicity analysis

---

**Input :** A set of paths  $cc$ , a solution OptS, a path  $pt_s$ .  
**Output:** The new (in case) optimal solution OptS.

```

if  $cc$  is empty then
    return OptS;
else
    OptSols  $\leftarrow \emptyset$ ;
    foreach  $pt_x \in cc$  do
        if ( $\exists pt_i \in OptS : pt_x \leftrightarrow pt_i$ ) and OptS is not empty then
            OptSols  $\leftarrow$  OptSols  $\cup$  OptS ;
        else
            tryOptS  $\leftarrow$  OptS  $\cup$  { $pt_x$ };
             $\tau \leftarrow$  getTau( $cc - \{pt_x\}$ ,  $pt_s$ );
            if score(tryOptS,  $Q$ )  $\geq \tau$  then
                OptSols  $\leftarrow$  OptSols  $\cup$  MonotonicityAnalysis( $cc - \{pt_x\}$ ,
                tryOptS,  $pt_s$ );
            else
                OptSols  $\leftarrow$  OptSols  $\cup$  OptS ;
    OptS  $\leftarrow$  TakeMaximal(OptSols) ;
    return OptS;
```

---

**Algorithm 8:** Take maximal

---

**Input :** A list OptimSols of solutions.  
**Output:** A solution best.

```

maxScore  $\leftarrow 0$ ;
maxCardinality  $\leftarrow 0$ ;
foreach sol  $\in$  OptimSols do
    if |sol|  $\geq$  maxCardinality then
        maxCardinality  $\leftarrow$  |sol|;
        if score(sol,  $Q$ )  $>$  maxScore then
            maxScore  $\leftarrow$  score(sol);
            best  $\leftarrow$  sol;
return best;
```

---

is  $p_4$ . The value of  $\tau$  is 1.86. Then the algorithm will retrieve the following admissible optimal solutions:  $S'_1(\{p_1, p_2, p_3\})$ ,  $S'_2(\{p_1, p_3\})$ , and  $S'_3(\{p_1, p_2, p_5\})$ . Such solutions are admissible because they satisfy the  $\tau$ -test and corresponding paths present pairwise intersections. During the computation, the analysis skips the solutions  $S'_4(\{p_1, p_2, p_3, p_5\})$  and  $S'_5(\{p_1, p_3, p_5\})$  because they do not satisfy the  $\tau$ -test: the scores of  $S'_4$  and  $S'_5$  are 1.55 and 1.26, respectively, which are both less than  $\tau$ . Finally, the function TakeMaximal will select  $S'_1$  as the final first optimal solution  $S_1$  since it has more sound paths and the highest score. Following a similar process, at the second iteration, the algorithm will return  $S_2(\{p_4, p_5\})$  with a lesser score than  $S_1$ . Exhaustivity and specificity are optimally balanced (i.e., we have all the keywords and strictly correlated).

508

509

510

511

512

513

514

515

516

517

518

Although such analysis reaches our target, the computational complexity of the building is in  $O(I^2)$ . As for Algorithm 2, in the worst case, the computation iterates  $k$  times. In lines [2–6], we follow the same strategy of Algorithm 2. Therefore, the executions in lines [4–5] and line [6] are in  $O(|\mathcal{CL}|) \in O(I)$  and  $O(I)$ , respectively. Then we have a conditional instruction: if the condition is true, we execute the monotonicity exploration (lines [8–10]); otherwise, we consider each connected component  $cc \in CC$  as a solution to insert into  $\mathcal{S}$  (lines [12–15]). As in Algorithm 2, the execution in lines [12–15] is in  $O(I)$ . In lines [8–10], we call the function `MonotonicityExploration` that executes the analysis of monotonicity at most  $I$  times. Such analysis is performed by the recursive function `MonotonicityAnalysis`: in Algorithm 7, the main executions are in lines [9–12] and line [15]. In both the execution is in  $O(I)$ , since we have  $I$  elements to analyze at the most. Since in Algorithm 6 both the operations in line [4] and in line [5] are in  $O(I)$ , the complexity of the monotonicity exploration is  $O(I^2)$ . Therefore, we conclude that the monotonic strategy to build top-k solutions is in  $O(I^2)$ .

To reduce the complexity of the monotonic strategy, we provide a variant of the monotonicity analysis, as shown in Algorithm 9. It allows to reach a linear time complexity of the overall process, that is,  $O(I)$ . Instead of computing all possible combinations between `OptS` and the paths  $pt_x \in cc$ , we select the top  $pt_x$  (one or more if equal score) having an intersection with a path of `OptS` (lines [4–7]). The task is performed by the function `DequeueWithIntersection`. The analysis will return `tryOptS` if it satisfies the  $\tau$ -test, `OptS` otherwise (lines [10–13]). The study of complexity is straightforward: in the worst case, in Algorithm 6, we have a unique connected component  $cc$  containing all input sound paths (i.e.,  $|cc| = I$ ). Therefore, the function `MonotonicityAnalysis` (the linear version) is invoked

---

**Algorithm 9:** Linear monotonicity analysis

---

**Input :** A priority queue  $cc$ , a solution `OptS`, a path  $pt_s$ .  
**Output:** A solution `OptS`.

```

if cc is empty then
    ↳ return OptS;
else
    if OptS is empty then
        ↳ ptx ← cc.Dequeue();
    else
        ↳ ptx ← cc.DequeueWithIntersection(OptS);
        tryOptS ← OptS ∪ {ptx};
        τ ← getTau(cc, pts);
        if score(tryOptS, Q) ≥ τ then
            ↳ return MonotonicityAnalysis(cc, tryOptS, pts);
        else
            ↳ cc ← cc ∪ {ptx};
    ↳ return OptS;

```

---

**Table 4.1** Top-k solutions building

Strategy	Complexity	Solution quality	
$\mathcal{L}$	$O(I)$	More $\mathcal{EX}$ rather than $\mathcal{SP}$	t9.1
$\mathcal{M}$	$O(I^2)$	Optimality between $\mathcal{EX}$ and $\mathcal{SP}$	t9.2
$\mathcal{LM}$	$O(I)$	More $\mathcal{SP}$ rather than $\mathcal{EX}$	t9.3

once: MonotonicityAnalysis recurses at most  $I$  times. The complexity of MonotonicityExploration is in  $O(I)$ , and then also the entire building by using the linear version of the monotonicity analysis is in  $O(I)$ . Recalling again our running example, we start from  $\text{First} = \{p_1, p_2, p_3, p_5\}$ . The optimal solution  $\text{OptS}$  starts from  $p_{t_1}$  having the highest score. Then we try to insert the second most relevant path, that is,  $p_2$ .  $\text{OptS} = \{p_1, p_2\}$  satisfies the  $\tau$ -test. Similarly, we can insert  $p_3$  into  $\text{OptS}$ , while we have to discard  $p_5$  since the  $\tau$ -test is not satisfied. Finally, we have the solution  $S_1(\{p_1, p_2, p_3\})$  and, at the second iteration,  $S_2(\{p_4, p_5\})$ . In this case, we obtain the same result but in linear time. Nevertheless, with respect to the building using Algorithm 7, we could generate solutions more specific and (possibly) less exhaustive, since we compose each solution starting from the most relevant sound path (i.e., we privilege strict connections between keywords). 557

Table 4.1 summarizes all discussed results. Given the size  $I$  of the input, that is, the number of sound paths, the linear strategy ( $\mathcal{L}$ ) shows a linear time complexity  $O(I)$  in the worst case, where the result shifts more to be exhaustive ( $\mathcal{EX}$ ) rather than specific ( $\mathcal{SP}$ ). The monotonic strategy ( $\mathcal{M}$ ) provides an optimal balance between  $\mathcal{EX}$  and  $\mathcal{SP}$ , with quadratic time complexity in the worst case, that is,  $O(I^2)$ . The variant of the monotonic strategy ( $\mathcal{LM}$ ) shows again a linear time complexity  $O(I)$ , where the result shifts more to be specific rather than exhaustive. 563

## 4.5 Related Work

This chapter is an evolution of a previous work [24] by the same authors. In that work, authors provided a path-based keyword search process to build top-k solutions in an approximate way: the focus was exclusively on the quality of the result, without addressing efficiency. In this chapter, we introduced the two strategies and, consequently, new algorithms to reach scalability and a formal checking of the monotonicity. 571

The most prominent proposals to keyword search can be classified in a *schema-based* or *schema-free* approach. Schema-based approaches [11, 17, 20] are common on relational databases. In such works, (top-k) answers are trees composed of joined tuples, so-called *joined tuples trees* (JTTs). The work in [11] is the first to incorporate an IR-style technique to build JTTs. The score of a solution results from the aggregation of the scores of the involved tuples, normalized by the size of the tree (i.e., number of tuples). The authors introduce an upper bound function that 578

bounds the score of potential answers to halt the top-k solution building as soon as 579 possible. However, because evaluating such function is expensive, authors resort to 580 an over-approximation of the upper bound, taking from accuracy. The work in [17] 581 is similar to [11], but candidate answers are seen as monolithic virtual documents 582 assessed by a standard IR function. Such technique avoids the need to design a 583 mechanism to merge the contributions from the several tuples and attributes in 584 the candidate answer. In [20], the authors provide three strategies to interconnect 585 tuples into JTTs: (1) all connected trees up to a certain size, (2) all sets of tuples 586 that are reachable from a root tuple within a radius, and (3) all sets of multicenter 587 subgraphs within a radius. A common drawback of these approaches is that when 588 executing a search, the keyword query is converted into a set of SQL statements 589 that should generate candidate answers. All the queries have to be executed, but 590 some (could) return empty results, leading to inefficiency, which is likely to worsen 591 with the size of the dataset. The approach in [25], still schema aware but working 592 on graphs, partially avoids this problem. Here the authors rely on an RDFS domain 593 knowledge to convert keywords in query guides that help users to incrementally 594 build the desired semantic query. While unnecessary queries are not built (thus not 595 executed), there is a strict dependency on the users' feedback. A second drawback is 596 the need to implement optimizations, in order to halt the process from computing all 597 possible answers before returning the top-k, which introduce approximations on the 598 top-k computation. Our approach overcomes all these limits. In the linear strategy, 599 while trading off accuracy in the top-k computation, we guarantee a system that 600 scales extremely well with the size of the input. In the monotonic strategy, top-k 601 solutions are computed in exact ranked order in the first k steps. 602

Schema-free approaches [1, 8, 10, 13–15, 23] are more general as they work on 603 arbitrary graph-shaped data. A general approach searches for the top-k connected 604 trees, each representing a (*minimal*) Steiner tree [7]. In [1, 13], the authors propose 605 backward and bidirectional graph exploration techniques, allowing polynomial data 606 complexity. In [14], assuming the size of the query bounded, authors prove that top- 607 k answers in an approximate order can be computed linearly to k and polynomially 608 in the size of the input. Inspired by [14], authors in [8] propose a practical 609 implementation of an engine returning answers in ranked order in polynomial 610 time. Specifically, answers are first generated in an approximate order, then ranked 611 more precisely in a second analysis. Another approximate approach is provided 612 in [15]. It precomputes and indexes all the maximal  $r$ -radius subgraphs on disk. 613 Then, extending the Steiner tree problem,  $r$ -radius Steiner graphs can be computed 614 by pruning undesired parts from the maximal r-radius subgraphs. In all of these 615 approaches, a relevant drawback is that finding a (*minimal*) Steiner tree is known 616 as an NP-hard problem [7]. Therefore, the algorithms rely on (rather) complex 617 sub-routines or heuristics to calculate approximations of Steiner trees. In the best 618 case, such proposals have polynomial complexity in time. A relevant result of our 619 framework is to present a polynomial time complexity (i.e., linear or quadratic) in 620 the worst case: the upper bound complexity of our approach is a lower bound for the 621 others. Moreover, while we can guarantee to build the top-k solutions in the first k 622 steps of the execution, these works can only provide approximate solutions or have 623

to compute all the solutions up front and then rank. In [10], authors present a bilevel index to expedite the discovery of connections between nodes matching keywords, in order to efficiently build the answer trees. A slightly different approach is the work in [23]. It proposes a system that interprets the keyword query into a set of candidate conjunctive queries. The user can refine the search by selecting which of the computed candidate queries best represent their information need. Candidate queries are computed exploring the top-k subgraphs matching the keywords. This is a semiautomatic system, opposed to ours that is completely automatic.

631

## 4.6 Experimental Results

632

We implemented our approach in a Java system. In our experiments, we used a widely accepted benchmark on DBLP<sup>3</sup> for keyword search evaluation. It is a dataset containing 26M triples about computer science publications. We used the same set of queries as in [10, 13, 23], and we compared our results with these works. Moreover, other two datasets were employed: IMDB<sup>4</sup> (a portion of 6M triples) and LUBM [9] (generating 50 universities corresponding to 8M triples). Experiments were conducted on a dual core 2.66GHz Intel Xeon, running Linux RedHat, with 4 GB of memory, 6 MB cache, and a 2-disk 1Tbyte striped RAID array, and we used Oracle 11g v2 as relational DBMS to build our index, as described below.

641

### 4.6.1 Index

642

To build solutions efficiently, we index the following information: vertices' labels (for keyword-to-element mapping) and the full-paths ending into sinks, since they bring information that might match the query. The first information enables to locate vertices matching the keywords, the second allows us to skip the expensive graph traversal at runtime. The indexing process is composed of three steps: (1) hashing of all vertices' labels, (2) identification of sources and sinks, and (3) computation of the full-paths. The first and the second steps are relatively easy. The third step requires to traverse the graph starting from the sources and following the routes to the sinks. To compute full-paths, we have implemented an optimized version of the breadth-first-search (BFS) paradigm, where independently concurrent traversals are started from each source. Similarly to [4], and differently from the majority of related works (e.g., [10]), we assume that the graph cannot fit in memory and that can only be stored on disk. Specifically, we store the index in Oracle, a relational DBMS.

Figure 4.2 shows the schema of our index, populated with the information from the graph of Fig. 4.1. Such schema is described in detail in [3]. Briefly, the table

<sup>3</sup><http://knoesis.wright.edu/library/ontologies/swetodblp/>

<sup>4</sup><http://www.linkedmdb.org/>

VERTICES		PATHS			PATHVERTICES			
VID	LABEL	PID	TID	F_VID	PID	VID	position	
01	aut1	23	13	06	pub1-type-Publication	28	05	1
02	aut2	24	13	06	pub2-type-Publication	28	07	2
03	aut3	25	14	10	pub1-year-2008	...	...	...
04	Researcher	26	14	10	pub2-year-2008	27	05	1
05	pub1	27	18	11	pub1-author-aut1-name-Bernstein	27	01	2
06	pub2	28	15	08	pub1-acceptedBy-conf1-type-Conference	27	12	3
07	Publication	29	16	09	pub1-acceptedBy-conf1-name-SIGMOD	...	...	...
08	conf1	30	17	03	pub1-author-aut1-type-Researcher			
09	Conference	31	18	12	pub2-author-aut2-name-Buneman			
10	SIGMOD	32	40	08	pub2-editedBy-conf1-type-Conference			
11	2008	33	50	09	pub2-editedBy-conf1-name-SIGMOD			
12	Bernstein	34	17	03	pub2-author-aut2-type-Researcher			
13	Buneman	35	18	30	pub2-author-aut3-name-Koltsidas			
14	Koltsidas							

TEMPLATES	
TID	template
15	#-type-#
16	#-year-#
17	#-acceptedBy #-type-#
18	#-acceptedBy #-name-#
19	#-author #-type-#
20	#-author #-name-#
21	#-editedBy #-type-#
22	#-editedBy #-name-#

Fig. 4.2 An example of logical modeling of the index

VERTICES describes information about vertices identified by an oid (i.e., VID that is the key of the table) corresponding to the hashing of the *label* (i.e., URI or literal values) and the relevance *rel*. The table PATHS contains the full-paths resulting from the BFS traversal: each path is identified by the oid *PID* and the corresponding sink *F\_VID* (an external reference to VERTICES) and presents the *length*, the relative relevance *rel*, and the associated template *TID*, as a reference to the table TEMPLATES containing information about templates of all occurring full-paths. Finally, PATHVERTICES has information of contained vertices for each path (e.g., necessary to test the intersection between paths). In particular, the table associates the *PID* of each path to the *VID* of the occurring vertices and the corresponding position of such vertices in the path (*PID* and *VID* represent the key). Because the index can be very large, in order to maintain a high level of performance, we fine-tune the relational DBMS at the physical level. Specifically, we employ the Oracle index-organized tables (IOTs) that are a special style of table structure stored in a BTree index frame. Along with primary key values, in the IOTs, we also store the nonkey column values. IOTs provide faster access to table rows by the primary key or any key that is a valid prefix of the primary key. Because the nonkey columns of a row are present in the BTree leaf block itself, there is no additional block access for index blocks. In our case, PATHS and PATHVERTICES are organized as IOTs. The matching is supported by standard IR engines (cf. Lucene domain index (LDi)<sup>5</sup>) embedded into Oracle as a type of index. In particular, we define an LDi index on the attributes *label* and *template* of tables VERTICES and TEMPLATES,

<sup>5</sup> [http://docs.google.com/View?id=ddgw7sjp\\_54fg9kg](http://docs.google.com/View?id=ddgw7sjp_54fg9kg)

respectively. Further, semantically similar entries such as synonyms, hyponyms, and hypernyms are extracted from WordNet [6], supported by LD<sub>i</sub>. In this way, from each input keyword, we provide a query on such tables, we extract the matching tuples, and following the references, we return the corresponding entries of PATHS. On top of this index organization, we implemented several procedures in PL/SQL, exploiting the native optimizations of Oracle, to support the maintenance: insertion, deletion, and update of new vertices or edges. The efficiency of such operations is documented in [3].

#### 4.6.2 Query Execution

689

For query execution evaluation, we compared the different strategies of our system (i.e., linear  $\mathcal{L}$ , monotonic  $\mathcal{M}$ , and the variant linear/monotonic  $\mathcal{LM}$ ), with the most related approaches: SEARCHWEBDB ( $SWDB$ ) [23], EASE ( $EASE$ ) [15], and the best performing techniques based on graph indexing, that is, 1,000  $BFS$  and 300  $BFS$  that are two configurations of BLINKS (see details in [10]). Table 4.2 shows the query set in detail. The table lists ten queries: for each one, we provide the list of keywords, and for each keyword, the number of matching and the corresponding number of retrieved sound paths to compose. For instance in  $Q_1$ , the keyword *algorithm* matches 1,299 nodes and retrieves about 31,590 sound paths.

We ran the queries ten times and measured the average response time. Precisely, the total time of each query is the time for computing the top ten answers. We performed *cold-cache* experiments (i.e., by dropping all file-system caches before restarting the various systems and running the queries): the query runtimes are shown in Fig. 4.3 (in *ms* and logarithmic scale).

In general,  $EASE$  and  $SWDB$  are comparable with  $BFS$ . Our system performs consistently better (in any strategy) for most of the queries, significantly

**Table 4.2** DBLP query dataset

Query	Input keywords	# of keyword nodes	# of sound paths	t10.1
$Q_1$	Algorithm 1999	(1299, 941)	(31590, 1071617)	t10.2
$Q_2$	Michael database	(744, 3294)	(44248, 15859)	t10.3
$Q_3$	Kevin statistical	(98, 335)	(5807, 4725)	t10.4
$Q_4$	Jagadish optimization	(1, 694)	(219, 11750)	t10.5
$Q_5$	Carrie carrier	(6, 6)	(174, 551)	t10.6
$Q_6$	Cachera cached	(1, 6)	(16, 69)	t10.7
$Q_7$	Jeff dynamic optimal	(45, 770, 579)	(2764, 16869, 10649)	t10.8
$Q_8$	Abiteboul adaptive algorithm	(1, 450, 1299)	(217, 14655, 31590)	t10.9
$Q_9$	Hector jagadish performance improving	(6, 1, 1349, 173)	(865, 219, 25631, 4816)	t10.10
$Q_{10}$	Kazutsgu johanna software performance	(1, 3, 993, 1349)	(4, 282, 43949, 25631)	t10.11

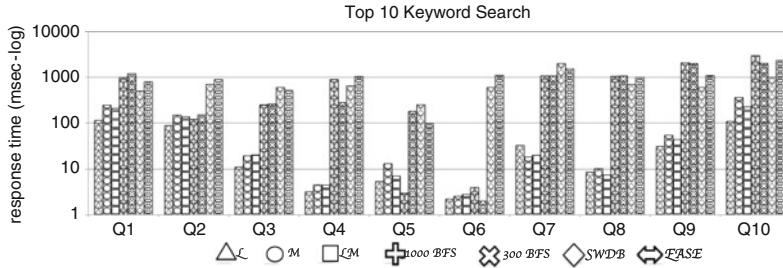


Fig. 4.3 Response times on DBLP

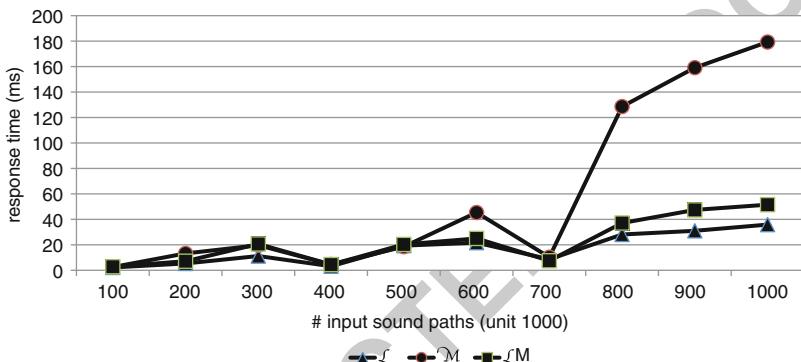


Fig. 4.4 Scalability with respect to #paths on DBLP

outperforming the others in some cases (e.g.,  $Q_4$  or  $Q_8$ ). This is due to the greatly reduced (time) complexity of the overall process with respect to the others that spend much time traversing the graph and computing candidates to be (in case) solutions. An evaluation of the scalability of our system is reported in Fig. 4.4. The figure shows the scalability with respect to the size of the input, that is, the number  $I$  of sound paths. It shows the search time (ms) for all ten queries with respect to the corresponding number of matching sound paths. The diagram proves the studies of complexity summarized in Table 4.1. Both  $\mathcal{L}$  and  $\mathcal{LM}$  follow a similar linear trend while  $\mathcal{M}$  shows a quadratic trend. We performed similar experiments on iMDB and LUBM. We formulated five queries for both datasets. Figure 4.5 shows the response times for our approach (in any strategy),  $\mathcal{EASE}$  and  $SWDB$ . The queries  $Q_1$  to  $Q_5$  were executed on iMDB while queries  $Q_6$  to  $Q_{10}$  on LUBM.  $Q_1$  and  $Q_5$  contain two keywords,  $Q_2$ ,  $Q_3$ ,  $Q_7$ ,  $Q_8$  three keywords, while the others four keywords. The results confirm the significant speedup of our approach with respect to the others.

We have also evaluated the effectiveness of results. The first measure we used is the reciprocal rank (RR). For a query, RR is the ratio between 1 and the rank at which the first correct answer is returned or 0 if no correct answer is returned. In DBLP, for all ten queries, we obtained  $RR = 1$  in  $\mathcal{M}$  and  $\mathcal{LM}$ . In this case, the

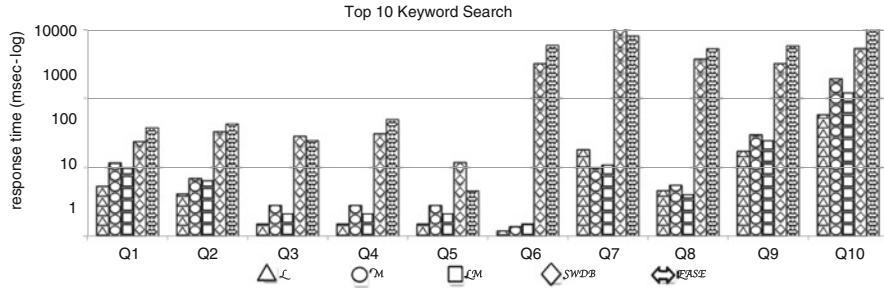


Fig. 4.5 Response times on iMDB and LUBM

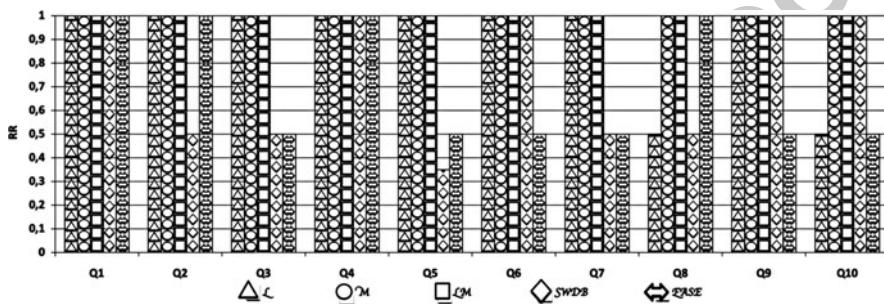


Fig. 4.6 RR measures for all frameworks in DBLP

monotonicity is never violated. In  $\mathcal{L}$ , we have an overall  $RR = 1$ , but  $RR = 0.5$  for  $Q_8$  and  $Q_{10}$  due to the violated monotonicity. As documented in Fig. 4.6, we compared our RR values, in any strategy, with that of  $\mathcal{EASE}$  and  $SWDB$ .

Then, we measured the interpolation between precision and recall that for each strategy on the DBLP dataset, that is, for each standard level  $r_j$  of recall (i.e.,  $0.1, \dots, 1.0$ ), we calculate the average max precision of queries in  $[r_j, r_{j+1}]$ , i.e.,  $P(r_j) = \max_{r_j \leq r \leq r_{j+1}} P(r)$ . We repeated this procedure for each strategy: Fig. ?? shows the results. As to be expected, the monotonic strategy  $\mathcal{M}$  presents the highest quality (i.e., a precision in the range  $[0.98, 1]$ ). Then the variant  $\mathcal{LM}$  and the linear strategy  $\mathcal{L}$  follow in order presenting good quality though. Such result confirms the feasibility of our system that produces the best solutions in linear time. The effectiveness on iMDB and LUBM follows a similar trend.

AQ5

## 4.7 Conclusion

736

In this chapter, we have presented a novel approach to keyword search query over large RDF datasets. We have provided two strategies for top-k query answering. The linear strategy enables the search to scale seamlessly with the size of the input,

737

738

739

producing solutions in an approximate order. The monotonic strategy guarantees the monotonicity of the output, that is, the first  $k$  solutions generated are indeed the top- $k$ . In the worst case, the two strategies present a linear and a quadratic computational cost, respectively, whereas other approaches show these results as lower bounds (i.e., best or average cases). Moreover, we described a variant of the second strategy that reaches both monotonicity and linear complexity. Experimental results confirmed our algorithms and the advantage over other approaches. This work opens several directions of further research. From a theoretical point of view, we are investigating a unique strategy that allows to reach both monotonicity and a linear time complexity, producing solutions that optimally balance exhaustivity and specificity. From a practical point of view, we are widening a more synthetic catalogue to index information, optimization techniques to speed up the index creation and update (mainly DBMS independent), and compression mechanisms.

## References

753

1. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword searching and browsing in databases using banks. ICDE, pp. 431–440 (2002) 754  
755
2. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Comput. Networks **30**(1–7), 107–117 (1998) 756  
757
3. Cappellari, P., Virgilio, R.D., Miscione, M., Roantree, M.: A path-oriented rdf index for keyword search query processing. DEXA (2011) 758  
759
4. Dalvi, B.B., Kshirsagar, M., Sudarshan, S.: Keyword search on external memory data graphs. Proc. VLDB **1**(1), 1189–1204 (2008) 760  
761
5. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. PODS, pp. 102–113 (2001) 762  
763
6. Fellbaum, C. (ed.): WordNet an Electronic Lexical Database. MIT, Cambridge, MA (1998) 764
7. Garey, M.R., Graham, R.L., Johnson, D.S.: The complexity of computing Steiner minimal trees. SIAM J. Appl. Math. **32**(4), 835–859 (1977) 765  
766
8. Golenberg, K., Kimelfeld, B., Sagiv, Y.: Keyword proximity search in complex data graphs. SIGMOD, pp. 927–940 (2008) 767  
768
9. Guo, Y., Pan, Z., Heflin, J.: Lubm: a benchmark for owl knowledge base systems. J. Web Sem. **3**(2–3), 158–182 (2005) 769  
770
10. He, H., Wang, H., Yang, J., Yu, P.S.: Blinks: ranked keyword searches on graphs. SIGMOD (2007) 771  
772
11. Hristidis, V., Gravano, L., Papakonstantinou, Y.: Efficient ir-style keyword search over relational databases. VLDB, pp. 850–861 (2003) 773  
774
12. Hristidis, V., Koudas, N., Papakonstantinou, Y., Srivastava, D.: Keyword proximity search in xml trees. IEEE Trans. Knowl. Data Eng. **18**(4), 525–539 (2006) 775  
776
13. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H.: Bidirectional expansion for keyword search on graph databases. VLDB (2005) 777  
778
14. Kimelfeld, B., Sagiv, Y.: Finding and approximating top- $k$  answers in keyword proximity search. PODS, pp. 173–182 (2006) 779  
780
15. Li, G., Ooi, B.C., Feng, J., Wang, J., Zhou, L.: Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. SIGMOD (2008) 781  
782
16. Liu, F., Yu, C.T., Meng, W., Chowdhury, A.: Effective keyword search in relational databases. SIGMOD (2006) 783  
784

17. Luo, Y., Lin, X., Wang, W., Zhou, X.: Spark: top-k keyword query in relational databases. SIGMOD (2007) 785  
786
18. Luo, Y., Wang, W., Lin, X., Zhou, X., Member, S., Wang, I.J., Li, K.: Spark2: Top-k keyword query in relational databases. IEEE Trans. Knowl. Data Eng. **99**, 1 (2011) 787  
788
19. Piwowarski, B., Dupret, G.: Evaluation in (xml) information retrieval: expected precision-recall with user modelling (eprum). SIGIR, pp. 260–267 (2006) 789  
790
20. Qin, L., Yu, J.X., Chang, L.: Keyword search in databases: the power of rdbms. SIGMOD (2009) 791  
792
21. Singhal, A.: Modern information retrieval: a brief overview. Data(base) Eng. Bull. **24**(4), 35–43 (2001) 793  
794
22. Singhal, A., Buckley, C., Mitra, M.: Pivoted document length normalization. SIGIR, pp. 21–29 (1996) 795  
796
23. Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. ICDE, pp. 405–416 (2009) 797  
798
24. Virgilio, R.D., Cappellari, P., Miscione, M.: Cluster-based exploration for effective keyword search over semantic datasets. ER, pp. 205–218 (2009) 799  
800
25. Zenz, G., Zhou, X., Minack, E., Siberski, W., Nejdl, W.: From keywords to semantic queries – incremental query construction on the semantic web. J. Web Semant. **7**(3), 166–176 (2009) 801  
802

AUTHOR QUERIES

- AQ1. Please confirm the changes in the sentence "...since data is usually scattered...".
- AQ2. Kindly check the corresponding author.
- AQ3. In the sentence "The length of a path..." 'into' has been changed as 'in'. Please check.
- AQ4. Please check if the sentence "Moreover, it cannot..." reads fine.
- AQ5. Kindly provide appropriate figure label here.

UNCORRECTED PROOF

# Chapter 5

## Interactive Query Construction for Keyword Search on the Semantic Web

1  
2  
3**Gideon Zenz, Xuan Zhou, Enrico Minack, Wolf Siberski, and Wolfgang Nejdl** 4

### 5.1 Introduction

5

With the advance of the semantic Web, increasing amounts of data are available 6 in a structured and machine-understandable form. This opens opportunities for 7 users to employ semantic queries instead of simple keyword-based ones to accu- 8 rately express the information need. However, constructing semantic queries is 9 a demanding task for human users [11]. To compose a valid semantic query, a 10 user has to (1) master a query language (e.g., SPARQL) and (2) acquire sufficient 11 knowledge about the ontology or the schema of the data source. While there are 12 systems which support this task with visual tools [21, 26] or natural language 13 interfaces [3, 13, 14, 18], the process of query construction can still be complex 14 and time consuming. According to [24], users prefer keyword search, and struggle 15 with the construction of semantic queries although being supported with a natural 16 language interface. 17

Several keyword search approaches have already been proposed to ease informa- 18 tion seeking on semantic data [16, 32, 35] or databases [1, 31]. However, keyword 19 queries lack the expressivity to precisely describe the user's intent. As a result, 20 ranking can at best put query intentions of the majority on top, making it impossible 21 to take the intentions of all users into consideration. 22

AQ1

---

G. Zenz (✉) · E. Minack · W. Siberski · W. Nejdl  
L3S Research Center, Appelstr. 9a, 30167 Hannover, Germany  
e-mail: [zenz@l3s.de](mailto:zenz@l3s.de); [minack@l3s.de](mailto:minack@l3s.de); [siberski@l3s.de](mailto:siberski@l3s.de); [nejdl@l3s.de](mailto:nejdl@l3s.de)

X. Zhou  
Renmin University of China, 500 Information BLD, Beijing 100872, China  
e-mail: [xuan.zhou.mail@gmail.com](mailto:xuan.zhou.mail@gmail.com)

In this chapter, we describe QUICK,<sup>1</sup> a system for querying semantic data. 23  
QUICK internally works on predefined domain-specific ontologies. A user starts 24  
by entering a keyword query, QUICK then guides the user through an incremental 25  
construction process, which quickly leads to the desired semantic query. Users are 26  
assumed to have basic domain knowledge, but do not need specific details of the 27  
ontology, or proficiency in a query language. In that way, QUICK combines the 28  
convenience of keyword search with the expressivity of semantic queries. This 29  
system has also been presented in [34]. 30

The chapter is organized as follows. Section 5.2 shows an introductory example 31  
to provide an overview on how to use QUICK. After introducing some preliminary 32  
notions in Sect. 5.3, Sect. 5.4 presents our framework for incremental query 33  
construction. In Sect. 5.6, we describe the algorithms for generating near-optimal 34  
query guides. Section 5.7 introduces optimization techniques to improve query 35  
execution performance. In Sect. 5.8, we present the results of our experimental 36  
evaluation. Section 5.9 reviews the related work. We close with conclusions in 37  
Sect. 5.10. 38

## 5.2 Motivating Example

Suppose a user looks for a movie set in *London* and directed by *Edgar Wright*.<sup>2</sup> The 40  
user starts by entering a keyword query, for instance “wright london.” Of course, 41  
these keywords can imply a lot of other semantic queries than the intended one. 42  
For example, one possible query is about an actor called *London Wright*. Another 43  
one could search for a character *Wright*, who was performed by an actor *London*. 44  
QUICK computes all possible semantic queries and presents selected ones in its 45  
interface (see right-hand side of the prototype interface shown in Fig. 5.1). 46

More importantly, it also generates a set of query construction options and 47  
presents them in the construction pane (left-hand side of the user interface). If the 48  
intended query is not yet offered, the user can incrementally construct this query by 49  
selecting an option in the construction pane. Whenever the user makes a selection, 50  
the query pane changes accordingly, zooming into the subset of semantic queries 51  
that conform to the chosen options. At the same time, a new set of construction 52  
options is generated and presented in the construction pane. We call this series of 53  
construction options *query guide*, because it offers the user a path to the intended 54  
query. In the screenshot, the user has already selected that “london” should occur 55  
in the movie plot, and is now presented alternate construction options for “wright.” 56  
When the user selects the desired query, QUICK executes it and shows the results. 57

The generated construction options ensure that the space of semantic interpre- 58  
tations is reduced rapidly with each selection. For instance, by specifying that 59

<sup>1</sup>QUery Intent Constructor for Keywords.

<sup>2</sup>Throughout this paper, we use the IMDB movie dataset as an example to illustrate our approach.

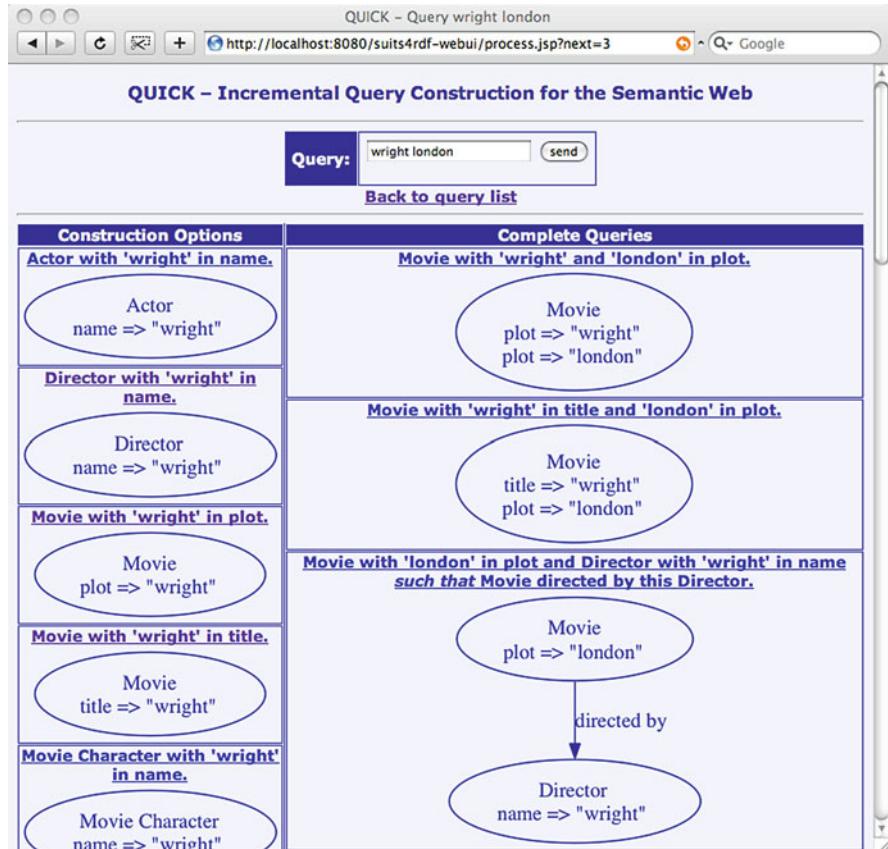


Fig. 5.1 QUICK user interface

"london" refers to a movie and not a person, more than half of all possible semantic 60 queries are eliminated. After a few choices, the query space comprises only a few 61 queries, from which the user can select the intended one easily. 62

In the following sections, we introduce the query construction framework and 63 show how the interactive construction process of semantic queries is supported in 64 QUICK. 65

### 5.3 Preliminaries

QUICK works on any RDF knowledge base with an associated schema in RDFS; 67 this schema is the basis for generating semantic queries. We model schema 68 information as a *schema graph*, where each node represents either a concept or a free 69

literal, and each edge represents a property by which two concepts are related. To 70  
keep Definition 5.1 simple, we assume explicit rdf:type declarations of all concepts. 71

**Definition 5.1 (Knowledge Base).** Let  $L$  be the set of literals,  $U$  the set of URIs. 72  
A knowledge base is a set of triples  $G \subset (U \times U \times (U \cup L))$ . We use  $R = 73$   
 $\{r \in U \mid \exists(s \ p \ o) \in G : (r = s \vee r = o)\}$  to represent the set of resources,<sup>3</sup> 74  
 $P = \{p \mid \exists s, o : (s \ p \ o) \in G\}$  to represent the set of properties, and  $C = \{c \mid \exists s : 75$   
 $(s \text{ rdf:type } c) \in G\}$  to represent the set of concepts. 76

**Definition 5.2 (Schema Graph).** The schema graph of a knowledge base  $G$  is 77  
represented by  $SG = (C, EC, EL)$ , where  $C$  denotes a set of concepts,  $EC$  denotes 78  
the possible relationships between concepts, and  $EL$  denotes possible relationships 79  
between concepts and literals. Namely,  $EC = \{(c_1, c_2, p) \mid \exists r_1, r_2 \in R, p \in P : 80$   
 $(r_1, p, r_2) \in G \wedge (r_1 \text{ rdf:type } c_1) \in G \wedge (r_2 \text{ rdf:type } c_2) \in G\}$ , and  $EL = 81$   
 $\{(c_1, p) \mid \exists r_1 \in R, p \in P, l \in L : (r_1, p, l) \in G \wedge (r_1 \text{ rdf:type } c_1) \in G\} 82$

The schema graph serves as the basis for computing query templates, which 83  
allow us to construct the space of semantic query interpretations, as discussed in 84  
the following. 85

## 5.4 From Keywords to Semantic Queries

86

When a KEYWORD QUERY  $kq = \{t_1, \dots, t_n\}$  is issued to a knowledge base, it 87  
can be interpreted in different ways. Each interpretation corresponds to a semantic 88  
query. For the query construction process, QUICK needs to generate the complete 89  
semantic query space, i.e., the set of all possible semantic queries for the given set 90  
of keywords. 91

The query generation process consists of two steps. First, possible query patterns 92  
for a given schema graph are identified, not taking into account actual keywords. We 93  
call these patterns *query templates*. Templates corresponding to the example queries 94  
from Sect. 5.2 are: “retrieve movies directed by a director” or “retrieve actors who 95  
have played a character.” 96

Formally, a query template is defined as composition of schema elements. To 97  
allow multiple occurrences of concepts or properties, they are mapped to unique 98  
names. On query execution, they are mapped back to the corresponding source 99  
concept/property names of the schema graph. 100

**Definition 5.3 (Query Template).** Given a schema graph  $SG = (C_{SG}, EC_{SG}, EL_{SG})$ , 101  
 $T = (C_T, EC_T, EL_T)$  is a query template of  $SG$ , iff (1) there is a function 102  
 $\tau : C_T \rightarrow C_{SG}$  mapping the concepts in  $C_T$  to the source concepts in  $C_{SG}$ , 103  
such that  $(c_1, c_2, p) \in EC_T \Rightarrow (\tau(c_1), \tau(c_2), p) \in EC_{SG}$  and  $(c_1, L, p) \in EL_T \Rightarrow 104$

---

<sup>3</sup>To keep the presentation clear we do not consider blank nodes; adding them to the model is straightforward.

$(\tau(c_1), L, p) \in EL_{SG}$ , and (2) the graph defined by  $T$  is connected and acyclic. We call a concept that is connected to exactly one other concept in  $T$  leaf concept.

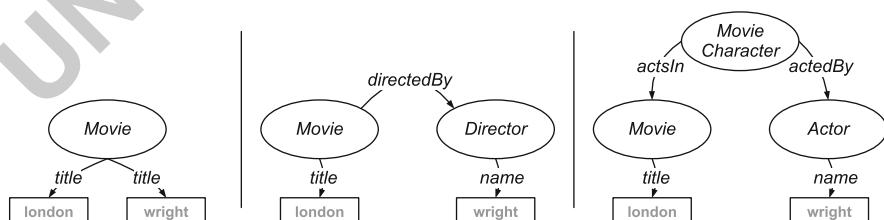
Figure 5.2 shows three query templates with sample variable bindings. QUICK automatically derives all possible templates off-line from the schema graph (up to a configurable maximum size), according to Definition 5.3. This is done by enumerating all templates having only one edge, and then recursively extending the produced ones by an additional edge, until the maximum template size is reached.

Currently, we limit the expressivity of templates to acyclic conjunctions of triple patterns. Further operators (e.g., disjunction) could be added, however, at the expense of an increased query space size. Investigation of this trade-off is part of future work.

In the second step, *semantic queries* are generated by binding keywords to query templates. A keyword can be bound to a literal if an instance of the underlying knowledge base supports it. Alternatively, it can be bound to a concept or property, if the keyword is a synonym (or homonym) of the concept or property name. A full-text index on the knowledge base is used to efficiently identify such bindings. Figure 5.2 shows some semantic queries for the keyword set “wright london,” which bind the keywords to the literals of three different query templates. The left one searches for a movie with “wright” and “london” in its title. The middle one searches for a movie with “london” in its title directed by a director “wright.” The right one searches for an actor “wright” playing a character in a movie with “london” in its title. Furthermore, keywords can also be matched to properties and classes, such as “name” or “Movie.”

**Definition 5.4 (Semantic Query).** Given a keyword query  $kq$ , a semantic query is a triple  $sq = (kq, T, \theta)$ , where  $T = (C_T, EC_T, EL_T)$  is a query template, and  $\theta$  is a function which maps  $kq$  to the literals, concepts, and properties in  $T$ .  $sq = (kq, T, \theta)$  is a valid semantic query, iff for any leaf concept  $c_i \in C_T$ , there exists a keyword  $k_i \in kq$  that is mapped by  $\theta$  to  $c_i$  itself, or a property or a literal connected to  $c_i$ .

The SEMANTIC QUERY SPACE for a given query  $kq$  and schema graph  $SG$  is the set of all queries  $SQ = \{sq | \exists \theta, T : (kq, T, \theta) \text{ is a query template}\}$ .



**Fig. 5.2** Sample query templates for the IMDB schema; the terms in gray represent instantiations of these templates to semantic queries for “wright london”

In our model, each term is bound separately to a node of a template. Phrases can be expressed by binding all corresponding terms to the same property, cf. the left-hand example in Fig. 5.2. Additionally, linguistic phrase detection could be performed as a separate analysis step; in this case, a phrase consisting of several keywords would be treated as one term when guiding the user through the construction process.

The QUICK user interface prototype shows the queries as graphs as well as in textual form. The query text is created by converting graph edges to phrases. For each edge connecting a concept with a bound property, we create the phrase “`<concept> with <keyword> in <property>`,” using the respective concept and property labels. If an edge connects two concepts, the relation is translated to the phrase “`this <concept1> <property> this <concept2>`.” For the second query in Fig. 5.2, the following text is generated: “*Movie with “london” in title and Director with “wright” in name such that Movie directed by this Director.*”

As shown in Sect. 5.7, a semantic query corresponds to a combination of SPARQL triple pattern expressions, which can be directly executed on an RDF store.

## 5.5 Construction Guides for Semantic Queries

QUICK presents the user with query construction options in each step. By selecting an option, the user restricts the query space accordingly.

These options are similar to semantic queries, except they do not bind all query terms. Therefore, the construction process can be seen as the stepwise process of selecting partial queries that subsume the intended semantic query.

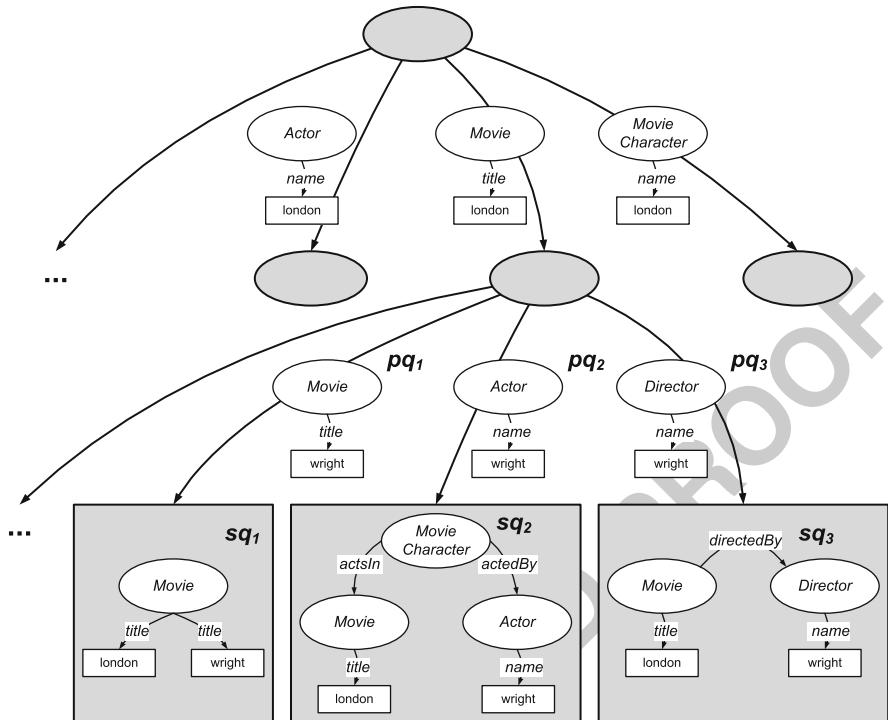
To describe precisely how a query guide is built, we introduce the notions of partial query and of subquery relationship. Our notion of query subsumption relies on the RDF Schema definition of concept and property subsumption. Note that our algorithms are not dependent on a specific definition of query subsumption, it would work equally well with more complex approaches, e.g., concept subsumption in OWL.

**Definition 5.5 (subquery, Partial Query).**  $sa = (q_a, T_a, \theta_a)$  is a subquery of  $sb = (q_b, T_b, \theta_b)$ , or  $sa$  subsumes  $sb$ , iff:

- 1.1  $q_a \subset q_b$ .
- 2.1 There exists a subgraph isomorphism  $\phi$  between  $T_a$  and  $T_b$ , so that each concept  $a_1 \in T_a$  subsumes  $\phi(a_1)$  and each property  $p \in T_a$  subsumes  $\phi(p)$ .
- 3.1 For any  $k_1 \in q_a$ ,  $\phi(\theta_a(k_1)) = \theta_b(k_1)$ .

A partial query is a subquery of a semantic query.

For example, in Fig. 5.3, the partial queries  $pq_1$ ,  $pq_2$ , and  $pq_3$  are subqueries of  $sq_1$ ,  $sq_2$ , and  $sq_3$ , respectively.



**Fig. 5.3** Part of a query guide for “wright london”

The construction options of QUICK are modeled as a *query construction graph* 174 (QCG), as illustrated in Fig. 5.3. While the example shown is a tree, in general, the 175 QCG can be any directed acyclic graph with exactly one root node. Given a set of 176 semantic queries  $SQ$ , a QCG of  $SQ$  satisfies: 177

- 1.1 The root of QCG represents the complete set of queries in  $SQ$ . 178
- 2.1 Each leaf node represents a single semantic query in  $SQ$ . 179
- 3.1 Each non-leaf node represents the union of the semantic queries of its children. 180
- 4.1 Each edge represents a partial query. 181
- 5.1 The partial query on an incoming edge of a node subsumes all the semantic 182 queries represented by that node. 183

**Definition 5.6 (Query Construction Graph).** Given a set of semantic query  $SQ$  184 and its partial queries  $PQ$ , a query construction graph is a graph  $QCG = (V, E)$ , 185 where  $V \subset SQ^*$ ,  $E \subset \{(v_1, v_2, p) | v_1, v_2 \in V, v_2 \subset v_1, p \in PQ, p \text{ subsumes } v_2\}$  186 and  $\forall v \in V, |v| > 1 : v = \{\bigcup v_i | \exists p : (v, v_i, p) \in E\}$ . 187

If  $SQ$  is the complete query space of a keyword query, a QCG of  $SQ$  is a *Query 188 Guide* of the keyword query. 189

**Definition 5.7 (Query Guide).** A *query guide* for a keyword query is a query construction graph whose root represents the complete semantic query space of that keyword query. 190  
191  
192

With a query guide, query construction can be conducted. The construction 193  
process starts at the root of the guide, and incrementally refines the user's intent 194  
by traversing a path of the graph until a semantic query on a leaf is reached. In each 195  
step, the user is presented the partial queries on the outgoing edges of the current 196  
node. By selecting a partial query, the user traverses to the respective node of the 197  
next level. In the example of Fig. 5.3, after having chosen that "wright" is part of 198  
the actor's name and "london" part of the movie's title, QUICK can already infer 199  
the intended query. The properties of the query construction graph guarantee that a 200  
user can construct every semantic query in the query space. 201

Every query guide comprises the whole query space, i.e., covers all query 202  
intentions. However, these guides vary widely in features such as branching factor 203  
and depth. A naïvely generated guide will either force the user to evaluate a huge list 204  
of partial query suggestions at each step (width), or to take many steps (depth) until 205  
the query intention is reached. For example, for only 64 semantic queries, a naïve 206  
algorithm produces a guide which requires the user to evaluate up to 100 selection 207  
options to arrive at the desired intention, while an optimal guide only requires 17. 208  
Therefore, we aim at a query guide with minimal interaction cost. 209

We define the interaction cost as the number of partial queries the user has to 210  
view and evaluate to finally obtain the desired semantic query. As QUICK does 211  
not know the intention when generating the guide, the worst case is assumed: the 212  
interaction cost is the cost of the path through the guide incurring most evaluations 213  
of partial queries. This leads to the following cost function definition. 214

**Definition 5.8 (Interaction Cost of a Query Construction Graph).** Let  $cp =$  215  
 $(V', E')$  be a path of a query construction graph  $QCG = (V, E)$ , i.e.,  $V' = \{v_1, \dots, v_n\} \subset V$  216  
 $\subset V$  and  $E' = \{(v_1, v_2, p_1), (v_2, v_3, p_2), \dots, (v_{n-1}, v_n, p_{n-1})\} \subset E$ . Then:  $cost(cp) = \sum_{p \in E'} |\{p : (v, v_1, p) \in E\}|$  and  $cost(QCG) = \max(cost(cp) : cp)$ . 217  
218

**Definition 5.9 (Minimum Query Construction Graph).** Given a set of semantic 219  
queries  $SQ$ , a query construction graph  $QCG$  is a *minimum query construction graph* 220  
of  $SQ$ , iff there does not exist another query construction graph  $QCG'$  of  $SQ$  such 221  
that  $cost(QCG) > cost(QCG')$ . 222

A query guide which satisfies Definition 5.9 leads the user to the intended query 223  
with minimal interactions. In the following section, we show how to compute such 224  
guides efficiently. 225

## 5.6 Query Guide Generation

226

For a given keyword query, multiple possible query guides exist. While every guide 227  
allows the user to obtain the wanted semantic query, they differ significantly in 228  
effectiveness as pointed out in Sect. 5.5. It is thus essential to find a guide that 229

imposes as little effort on the user as possible, i.e., a minimum query guide. Query construction graphs have several helpful properties for constructing query guides: 230  
231

### Lemma 5.1 (Query Construction Graph Properties) 232

- (i) Given a node in a query construction graph, the complete subgraph with this node as root is also a query construction graph. 233  
234
- (ii) Suppose QCG is a query construction graph, and A is the set of children of its root. The cost of QCG is the sum of the number of nodes in A and the maximum cost of the subgraphs with root in A, i.e.,  $\text{Cost}(T) = |A| + \text{MAX}(\text{Cost}(a) : a \in A)$ . 235  
236  
237  
238
- (iii) Suppose QCG is a minimum query construction graph and A is the set of children of its root. If g is the most expensive subgraph with root in A, then g is also a minimum query construction graph. 239  
240  
241

### 5.6.1 Straightforward Guide Generation 242

Lemma 5.1 can be exploited to construct a query construction guide recursively. 243  
Based on property (ii), to minimize the cost of a query construction graph, we need 244  
to minimize the sum of (1) the number of children of the root, i.e.,  $|A|$  and (2) 245  
the cost of the most expensive subgraph having one of these children as root, i.e., 246  
 $\text{MAX}(\text{Cost}(a) : a \in A)$ . Therefore, to find a minimum construction graph, we can 247  
first compute all its possible minimum subgraphs. Using these subgraphs, we find a 248  
subset A with the minimum  $|A| + \text{MAX}(\text{Cost}(a) : a \in A)$ . The method is outlined 249  
in Algorithm 10. 250

---

#### Algorithm 10: Straightforward query guide generation

---

```

Simple_QGuide()
Input: Partial queries P, semantic queries S
Output: Query guide G.
if  $|S| = 1$  then
    return S ;
for each  $p \in P$  do
     $p.sg := \text{Simple\_QGuide}(P - \{p\}, S \cap p.SQ)$  ;
    //  $p.SQ$  denotes the semantic queries subsumed by p
     $G.cost := \infty$  ;
for each  $p \in P$  do
     $Q(p) := \{(p' \in P) : p'.sg.cost \leq p.sg.cost\}$  ;
     $min\_set := \text{MINSETCOVER}(S - p.SQ, Q(p))$  ;
    if  $G.cost > |min\_set| + p.sg.cost + 1$  then
         $G := min\_set \cup \{p\}$  ;
         $G.cost := |min\_set| + p.sg.cost$  ;
return G;
```

---

According to Lemma 5.1, this algorithm always finds a minimum query guide. 251  
 However, it relies on the solution of the SETCOVER problem, which is NP complete 252  
 [12]. Although there are polynomial approximation algorithms for MINSETCOVER 253  
 with logarithmic approximation ratio, our straightforward algorithm still incurs 254  
 prohibitive costs. Using a greedy MINSETCOVER, the complexity is still  $O(|P|! \cdot 255$   
 $|P|^2 \cdot |S|)$ . In fact, we can prove that the problem of finding a minimum query guide 256  
 is NP hard. 257

**Definition 5.10 (minSetCover).** Given a universe  $U$  and a family  $S$  of subsets 258  
 of  $U$ , find the smallest subfamily  $C \subset S$  of sets whose union is  $U$ . 259

**Theorem 5.1.** *The MINCONSTRUCTIONGRAPH problem is NP hard.* 260

*Proof.* We reduce MINSETCOVER to MINCONSTRUCTIONGRAPH: 261  
 $M_S : U \leftrightarrow U_S$  is a bipartite mapping between  $U$  and a set of semantic queries  $U_S$ . 262  
 $M_P : S \leftrightarrow S_P$  is a bipartite mapping between  $S$  and a set of partial queries  $S_P$ , 263  
 such that each partial query  $p \in S_P$  subsumes the semantic queries  $M_S(M_P^{-1}(p))$ . 264  
 Create another set of semantic queries  $A_S$  and a set of partial queries  $A_P$ . Let  $|A_S| = 265$   
 $2 \times |M_S|$ . Let  $A_P$  contain two partial queries, each covering half of  $A_S$ . Therefore, 266  
 the cost of the minimum query construction graph of  $A_S$  is  $|M_S| + 1$ , which is 267  
 larger than any query construction graph of  $M_S$ . Based on Lemma 5.1, if we solve 268  
 MINCONSTRUCTIONGRAPH( $U_S \cup A_S, U_P \cup A_P$ ), we solve MINSETCOVER( $U, S$ ). 269

### 5.6.2 Incremental Greedy Query Guide Generation

270

As shown, the straightforward algorithm is too expensive. In this section, we 271  
 propose a greedy algorithm, which computes the query construction graph in a top- 272  
 down incremental fashion. 273

Algorithm 11 starts from the root node and estimates the optimal set of partial 274  
 queries that cover all semantic queries. These form the second level of the query 275  
 construction graph. In the same fashion, it recurses through the descendants of the 276  
 root to expand the graph. Thereby, we can avoid constructing the complete query 277  
 construction graph; as the user refines the query step by step, it is sufficient to 278  
 compute only the partial queries of the node the user has just reached. 279

The algorithm selects partial queries one by one. Then, it enumerates all 280  
 remaining partial queries and chooses the one incurring minimal *total estimated* 281  
*cost*. It stops when all semantic queries are covered. The complexity of the algorithm 282  
 is  $O(|P| \cdot |S|)$ . 283

The formula for the *total estimated cost* of a query construction graph is given in 284  
 Definition 5.11. 285

**Definition 5.11 (Total Estimated Cost).** Let  $S$  be the semantic queries to cover, 286  
 $SP$ , the set of already selected partial queries, and  $p$ , the partial query to evaluate, 287  
 the estimated cost of the cheapest query construction graph is 288

**Algorithm 11:** Incremental greedy query guide generation

---

**Input:** partial queries  $P$ , semantic queries  $S$   
**Output:** query guide  $G$

```

if  $|S| = 1$  then
     $\sqcup$  return  $S$  ;
 $G := \emptyset$ 
while  $|S| \neq 0$  do
    select  $p \in P$  with min.  $TotalEstCost(S, G \cup \{p\})$ 
    if no such  $p$  exists then
         $\sqcup$  break ;
         $G := G \cup \{p\}$  ;
         $S := S - p.SQ$  ;
        //  $p.SQ$  denotes the semantic queries subsumed by  $p$ 
    if  $|S| \neq 0$  then
         $\sqcup$   $G := G \cup S$  ;
return  $G$ 
```

---

$$TotalEstCost(S, SP) = |S| \frac{|SP|}{|S \cap \bigcup SP|} + \max(minGraphCost(|p|) : p \in SP), \quad 289$$

where 290

$$minGraphCost(n) = \begin{cases} n = 1 : & 0 \\ n = 2 : & 2 \\ n > 2 : & e \cdot \ln(n) \end{cases} \quad 291$$

Here,  $minGraphCost(|p|)$  estimates the minimum cost of the query construction graph of  $p$ . Suppose  $f$  is the average fan-out for  $n$  queries, then the cost is approximately  $f \cdot \log_f(n)$ , which is minimal for  $f = e$ . The first addend of  $TotalEstCost$  estimates the expected number of partial queries that will be used to cover all semantic queries. This assumes the average number of semantic queries covered by each partial query does not vary. 292  
293  
294  
295  
296  
297

As discussed above, the algorithm runs in polynomial time with respect to the number of partial and semantic queries. Although the greedy algorithm can still be costly when keyword queries are very long, our experimental results of Sect. 5.8.4 show that it performs very well if the number of keywords is within realistic limits. 298  
299  
300  
301

## 5.7 Query Evaluation

302

When the user finally selects a query that reflects the actual intention, it will be converted to a SPARQL query and evaluated against an RDF store to retrieve the results. The conversion process is straightforward: for each concept node in the query or edge between nodes, a triple pattern expression of SPARQL is generated. In the first case, it specifies the node type; in the second case, it specifies the relation 303  
304  
305  
306  
307

```

SELECT * WHERE { ?movie rdf:type imdb:Movie.
    ?movie rdf:type imdb:Director.
    ?movie imdb:directedBy ?director.
    ?movie imdb:title ?term1 FILTER regex(?term1, ".*london.*").
    ?director imdb:name ?term2 FILTER regex(?term2, ".*wright.*"). }

```

**Fig. 5.4** SPARQL query for the right-hand sample in Fig. 5.2

between the nodes. Finally, for each search term, a filter expression is added. See Fig. 5.4 for an example.<sup>4</sup>

To make QUICK work, these SPARQL queries need to be processed efficiently. Recent research in the area of SPARQL query evaluation has made significant progress [23, 25, 28, 30], mainly based on dynamic programming, histograms, and statistical methods for estimating cardinality and selectivity. However, available mature SPAQL engines either do not employ these techniques or do not offer them in combination with the full-text querying capabilities QUICK requires. In [20], we evaluated the performance of the most popular RDF engines with respect to these features, using an extended Lehigh University Benchmark (LUBM). The results show that the evaluated engines do not perform well on the type of hybrid queries needed by QUICK, as they still use rather simple heuristics to optimize the order of join operations.

Statistics of join cardinality can be employed to optimize the join order effectively. We adopted this approach in QUICK to improve the semantic query execution performance. In the following, we show how to estimate the cardinality of each triple pattern, and explain how to exploit these estimates for join ordering.

### 5.7.1 Cardinality Estimation

Triple patterns in SPARQL queries can be categorized into four types. For each type, we use a different method to estimate the cardinality. Let  $s \ p \ o$  be a triple pattern, where  $p$  is always a constant. For patterns of type  $s \ p \ ?o$ , where  $!s$  is a constant and  $?o$  is a free variable, we exploit the BTree subject index, which is employed in most RDF stores [4, 9, 23]. We estimate the cardinality as the distance between the leftmost and rightmost matching element in the BTree. For patterns of type  $?s \ p \ o$ , the BTree object index is used. For patterns of type  $!s \ p \ ?o$ , where  $s$  is a variable already bound in the query plan, and  $o$  is a free variable, the cardinality can—assuming uniform distribution of values—be estimated as  $card(s, p) = |triples(p)| / |subjects(p)|$ , where  $triples(p)$  are the triples containing  $p$ , and  $subjects(p)$  are the distinct subjects occurring in these triples. The same

<sup>4</sup>In our actual implementation, we use the custom language extension of LuceneSail to express full-text filters.

technique is applied to the pattern  $?s \ p \ !o$ . These statistics can be precomputed 337  
and only need to be updated on significant knowledge base changes. 338

### 5.7.2 Join Order Optimization

339

Among the set of SPARQL algebra operators, i.e., *select*, *join*, and *triple pattern*, 340  
query execution costs are dominated by *join* in most cases. This makes join ordering 341  
crucial for efficient query processing. For a query containing  $n$  join operators, they 342  
cannot be exhaustively checked, as the number of possible join orders is factorial. 343  
Dynamic programming is thus used to identify a near-optimal join order [27]. Our 344  
implementation works as follows: It starts with a pool of (distinct) join sequence 345  
candidates, where each one initially contains only one join. Then, the join sequence 346  
with the minimum cost is chosen, and the join with smallest cardinality that is not 347  
yet in this sequence is added to the pool. The plan generation is complete as soon as 348  
the join sequence contains all joins. 349

To compute the join operator cost, we rely on the fact that RDF stores use the 350  
*Index Nested Loop Join* (INLJ) (e.g., [4, 10]). Let  $INLJ$  be an index nested loop 351  
join. Let  $INLJ.left$  ( $INLJ.right$ ) represent the left (right) child of the join. Then the 352  
join's cost can be calculated as  $cost(INLJ) = cost(INLJ.left) + (card(INLJ) * 353  
cost(INLJ.right))$ , where  $card(INLJ)$  represents the cardinality estimation of 354  
the join. The join cardinality is estimated as  $card(join) = card(join.left) * 355  
card(join.right)$ , where  $card(join.left)$  and  $card(join.right)$  are obtained using our 356  
statistics. 357

As shown in Sect. 5.8.4.1, this approach ensures efficient evaluation of semantic 358  
queries. 359

## 5.8 Experimental Evaluation

360

We implemented the QUICK system using Java. The implementation uses Sesame2 361  
[4] as RDF store and the inverted index provided by LuceneSail [19] to facilitate 362  
semantic query generation. Parts of the described query optimization approaches 363  
have been integrated to Sesame2 version 2.2. We have used this implementation 364  
to conduct a set of experiments to evaluate the effectiveness and efficiency of the 365  
QUICK system and present our results in this section. 366

### 5.8.1 Experiment Setup

367

Our experiments use two real-world datasets. The first one is the Internet Movie 368  
Database (IMDB). It contains 5 concepts, 10 properties, more than 10 million 369

instances, and 40 million facts. The second dataset (Lyrics, [6]) contains songs and artists, and consists of 3 concepts, 6 properties, 200,000 instances, and 750,000 facts. Although the vocabulary of the datasets is rather small, they still enable us to show the effectiveness of QUICK in constructing domain-specific semantic queries.

To estimate the performance of QUICK in real-world settings, we used a query log of the AOL search engine. We pruned the queries by their visited URLs to obtain 3,000 sample keyword queries for IMDB and 3,000 sample keyword queries for Lyrics Web pages. Most of these queries are rather simple, i.e., only referring to a single concept, such as a movie title or an artist's name, and thus cannot fully reflect the advantages of semantic queries. We therefore manually went through these queries and selected the ones referring to more than two concepts. This yielded 100 queries for IMDB and 75 queries for Lyrics, consisting of two to five keywords. We assume that every user has had a clear intent of the keyword query, implying that each one can be interpreted as a unique semantic query for the knowledge base. We manually assessed the intent and chose the corresponding semantic query as its interpretation. It turned out that most keyword queries had a very clear semantics. These interpretations served as the ground truth for our evaluation.

The experiments were conducted on a 3.60 GHz Intel Xeon server. Throughout the evaluation, QUICK used less than 1 GB memory.

### 5.8.2 Effectiveness of Query Construction

Our first set of experiments is intended to assess the effectiveness of QUICK in constructing semantic queries, i.e., how fast a user can turn a keyword query into the corresponding semantic query. At each round of the experiment, we issued a keyword query to QUICK and followed its guidance to construct the corresponding semantic query. We measured the effectiveness using the following two metrics:

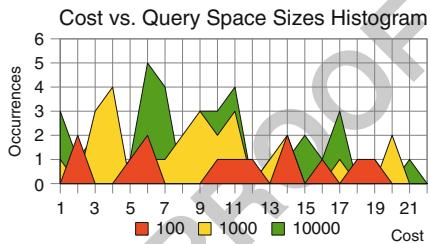
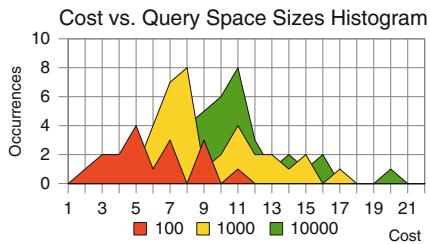
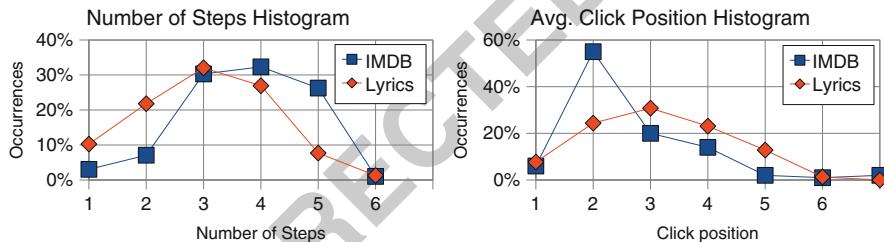
1. The interaction cost of each query construction process, i.e., the total number of options a user had evaluated to construct each semantic query
2. The number of selections a user performed to construct each semantic query, i.e., the number of clicks a user had to make

The results of the experiments are presented in Table 5.1 and Figs. 5.5 and 5.6.

Table 5.1 shows that the size of the semantic query space grows very fast with the number of terms in a keyword query. Because the datasets are large, a term usually occurs in more than one place of the schema graph. As the size of the query space is usually proportional to the occurrences of each term in the schema graph, it grows exponentially with the number of terms. Furthermore, even for less than five terms, the size of the query space can be up to 9,000 for IMDB and up to 12,000 for Lyrics—such a huge query space makes it difficult for any ranking function to work effectively. For comparison purposes, we applied the SPARK [35] ranking scheme to the semantic queries generated by QUICK, but experiments showed that SPARK could not handle it in a satisfactory manner. In most cases, a user needed to

**Table 5.1** Effectiveness of QUICK for IMDB (left) and Lyrics (right)

No. of terms	Avg. query space size	Cost		No. of terms	Avg. query space size	Cost	
		Avg.	Max			Avg.	Max
2	60.5	5.08	7	2	23.9	3.21	7
3	741	8.69	15	3	235	7.96	13
4	7,365	10.17	32	4	1,882	12.30	19
>4	9,449	14.88	33	>4	12,896	14.44	22
All	4,535	9.46	33	all	2,649	7.50	22

**Fig. 5.5** Query construction cost histograms for IMDB (left) and Lyrics (right) for three different query space sizes**Fig. 5.6** Histograms of the number of interactions and average click position

go through hundreds or thousands of queries to obtain the desired one. In contrast, QUICK displays steady performance when confronted with such big query spaces. As shown in Table 5.1, the maximum number of options a user needs to examine until obtaining the desired semantic query is always low (33 for IMDB rsp. 22 for Lyrics). On average, only 9 rsp. 7 options have to be examined by the user. The cost of the query construction process grows only linearly with the size of the keyword queries. This verifies our expectation that QUICK helps users in reducing the query space exponentially, enabling them to quickly construct the desired query.

Figure 5.5 shows the cost distribution of the query construction for different query space sizes. We can see that for most queries, the user only has to inspect between 4 and 11 queries. Only in rare cases, more than 20 queries had to be checked. The cost of the query construction process shows a similar trend, growing only logarithmically with the size of the query space. As shown on the left-hand side of Fig. 5.6, in most cases, only two to five user interactions were needed. On

AQ2

the right, we show the average position of the selected partial queries. These were 424  
almost always among the first five presented options. 425

To summarize, this set of experiments shows that QUICK effectively helps users 426  
to construct semantic queries. In particular, the query construction process enables 427  
users to reduce the semantic query space exponentially. This indicates the potential 428  
of QUICK in handling large-scale knowledge bases. 429

### 5.8.3 Efficiency

To demonstrate that QUICK is feasible for real-world applications, we conducted 431  
experiments to measure the computation time for generating query guides and for 432  
retrieving the final intended semantic queries. 433

#### 5.8.3.1 Query Guide Generation

We recorded the response times for each interaction of the query construction 435  
process. The initialization time as shown in the second and third columns of 436  
Table 5.2 comprises (1) the creation of the semantic query space, (2) computation of 437  
subquery relationships, and (3) keyword lookup in the full-text index. These tasks 438  
are fully dependent on the RDF store. The response times to user interactions were 439  
very short (fourth and fifth columns), enabling a smooth construction process. 440

In implementing QUICK, we focused on efficient algorithms for query guide 441  
generation and query evaluation. We are confident that for the initialization tasks, 442  
the performance can be improved significantly, too, e.g., by adapting techniques 443  
from [17] and by introducing special indexes. 444

#### 5.8.4 Quality of the Greedy Approach

To evaluate the quality of the query guides generated by the greedy algorithm 446  
discussed in Sect. 5.6.2, we compared it against the straightforward algorithm 447  
of Sect. 5.6.1. As the latter is too expensive to be applied to a real dataset, we 448

**Table 5.2** QUICK response time for IMDB (left) and lyrics (right)

No. of terms	Init time (ms)		Response time (ms)		t13.1
	IMDB	Lyrics	IMDB	Lyrics	
2	98	664	2	0.5	t13.3
3	993	384	19	4	t13.4
4	16,797	4,313	1,035	107	t13.5
>4	31,838	120,780	3,290	7,895	t13.6
All	3,659	17,277	314	1,099	t13.7

restricted the experiments to simulation. We generated artificial semantic queries, 449 partial queries, and subquery relationships. The semantic queries subsumed by each 450 partial query were randomly picked, while the number of these was fixed. Therefore, 451 three parameters are tunable when generating the queries,  $n\_complete$ —the number 452 of semantic queries;  $n\_partial$ —the number of partial queries; and  $coverage$ —the 453 number of semantic queries subsumed by each partial query. 454

In the first set of experiments, we fixed  $n\_complete$  to 128 and  $coverage$  to 48, 455 and varied  $n\_partial$  between 4 and 64. We run both algorithms on the generated 456 queries, and recorded the cost of the resulting query guides and their computation 457 time. To achieve consistent results, we repeatedly executed the simulation and 458 calculated the average. 459

As shown in the left-hand side of Fig. 5.7, the chance to generate cheaper 460 query guides increases with the number of partial queries. Guides generated by the 461 greedy algorithm are only slightly worse (by around 10%) than those generated by 462 the straightforward algorithm, independent of the number of partial queries. The 463 computation time of the straightforward algorithm increases exponentially with the 464 number of partial queries, while that of the greedy algorithm remains almost linear. 465 This is consistent with the complexity analysis of Sect. 5.6. 466

In the second set of experiments, we varied  $n\_complete$  between 32 and 256, 467 fixed  $n\_partial$  to 32, and set  $coverage$  to  $\frac{1}{4}$  of  $n\_complete$ . The results are shown 468 in Fig. 5.8. 469

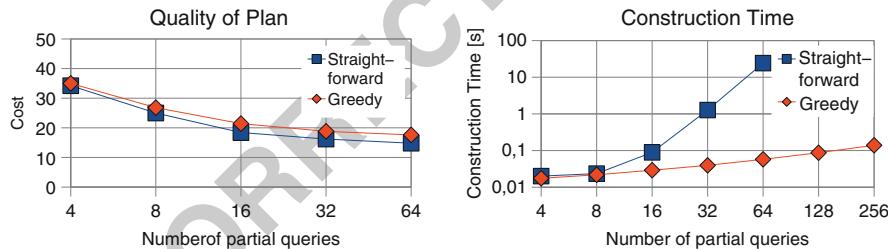


Fig. 5.7 Varying number of partial queries

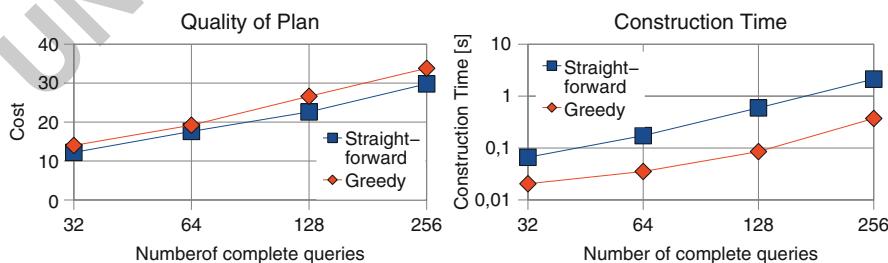
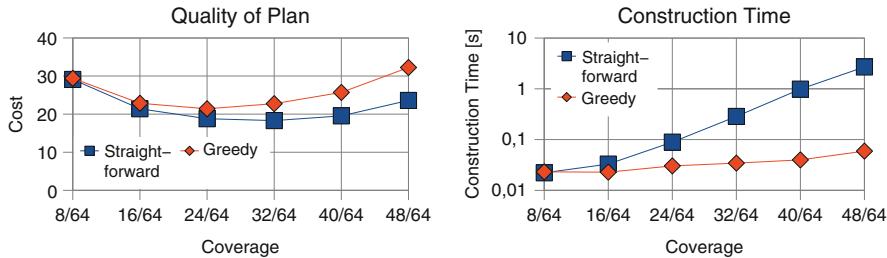


Fig. 5.8 Varying number of semantic queries



**Fig. 5.9** Varying coverage of partial queries

As expected, as the number of semantic queries increases exponentially, the cost of query construction increases only linearly. The guides generated by the greedy algorithm are still only slightly worse (by around 10%) than those generated by the straightforward algorithm. This difference does not change significantly with the number of semantic queries. The performance conforms to the complexity analysis of Sect. 5.6.

In the third set of experiments, we fixed  $n\_complete$  to 64 and  $n\_partial$  to 16, and varied  $coverage$  between 8 and 48.

Figure 5.9 shows that as the  $coverage$  increases, the cost of resulting query guides first decreases and then increases again. This confirms that partial queries with an intermediate coverage are more suitable for creating query construction graphs, as they tend to minimize the fan-out and the cost of the most expensive subgraph simultaneously. The difference between the greedy algorithm and the straightforward algorithm increases with the  $coverage$ . This indicates that the greedy algorithm has a nonconstant performance with respect to  $coverage$ , which was to be expected, as a result of the logarithmic performance rate of MINSETCOVER and the assumptions of Definition 5.11. Fortunately, in real data, most partial queries have a relatively small coverage (less than 20%), where this effect is less noticeable, justifying our assumptions.

In summary, the experiments showed the greedy algorithm to have the desired properties. In comparison to the straightforward algorithm, the generated guides are just slightly more costly for the user, but are generated much faster, thereby demonstrating the applicability of the QUICK approach.

#### 5.8.4.1 Query Evaluation

493

To assess the impact of the query evaluation techniques discussed in Sect. 5.7, we randomly selected 14 keyword queries from the IMDB query set. Using QUICK, we generated the semantic query space for these queries with a maximum size of 7. The 5,305 semantic queries which returned more than 100 results were used to assess the query evaluation performance.

We computed the improvement factor in terms of query acceleration. Figure 5.10 shows the distribution of this factor. In comparison with the unoptimized RDF store,

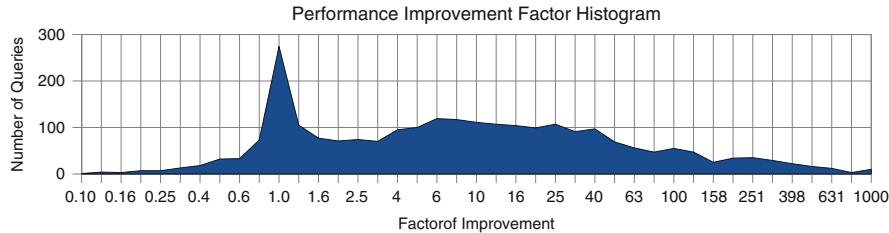


Fig. 5.10 Histogram of performance improvements

some queries did not run faster or even took slightly longer. However, the majority 501  
of the queries improved significantly (factor 4–40), demonstrating the benefits of 502  
our join order optimizations. 503

## 5.9 Related Work

In recent years, a number of user interfaces have been proposed to facilitate 505  
construction of semantic queries. These interfaces can be mainly classified into 506  
visual graphic interfaces [21, 26] and natural language interfaces [3, 14, 18]. Natural 507  
language interfaces are potentially more convenient for end users [13], as they 508  
require little prior knowledge of the data schema or a query language. However, the 509  
state-of-the-art natural language interfaces still require users to use a terminology 510  
that is compatible with the data schema and form grammatically well-formed 511  
sentences [13]. To overcome this limitation, a number of techniques have been 512  
used to help users construct valid natural language queries. For instance, the 513  
interface of [14] provides a dialog for users to align natural language terms to 514  
the ontology. The interface of [3] utilizes autocompletion to guide users to create 515  
meaningful sentences. In comparison with these methods, QUICK offers more 516  
flexibility in issuing queries. It accepts arbitrary keyword queries and allows users 517  
to incrementally structure the query through an optimized process. 518

Extensive investigations have recently been conducted on how to perform 519  
keyword search on the semantic Web [16, 32, 33, 35] and databases [1, 31]. The 520  
majority of work treats each dataset as a graph, and regards the query results as 521  
subgraphs, e.g., Steiner trees [15], which contain the query terms. They have in 522  
common that relevance ranking of these graphs is performed. As identifying the 523  
top-k query results would incur prohibitive I/O and computational cost, most of the 524  
approaches have been focusing on improving the performance of query processing. 525  
In contrast, QUICK is designed to help users to construct semantic queries. As the 526  
query results are retrieved after the semantic query is constructed, query processing 527  
is not a critical issue for QUICK. Work that is closest to QUICK includes [31, 32, 35]. 528  
Instead of retrieving search results directly, these approaches attempt to translate a 529  
keyword query into a set of semantic queries that are most likely to be intended by 530

users. However, these approaches can at best rank the most popular query intentions 531  
on top. As the number of semantic query interpretations is very high, it is impossible 532  
to take the intentions of all users into consideration. As QUICK allows users to 533  
clarify the intent of their keyword queries, it can satisfy diverse user information 534  
needs to a much higher degree. 535

Automatic query completion [2, 8, 22] and refinement [29] are techniques that 536  
help users form appropriate structured queries by suggesting possible structures, 537  
terms, or refinement options based on the partial queries the user has already 538  
entered. By using these suggestions, the user constructs correct database queries 539  
without completely knowing the schema. However, as this technique still requires 540  
users to form complete structured queries, it is less flexible than QUICK, which 541  
allows users to start with arbitrary keyword queries. Moreover, they do not tackle 542  
the issue of minimizing the users' interaction effort. 543

A main advantage of QUICK is its ability to allow users to clarify the intent 544  
of their keyword queries through a sequence of steps. In the area of informa- 545  
tion retrieval, document clustering and classification have been used for similar 546  
purposes, especially in connection with faceted search interfaces. For example, 547  
AQ3 Broughton and Heather [5] classify document sets based on predefined categories 548  
and let users disambiguate their search intent by selecting the most preferred 549  
categories. However, information retrieval approaches do not work on complex 550  
structured data, because joins over different concepts are not supported. 551

In [7], the idea of interactive query construction was combined with a probabilis- 552  
tic model for the possible informational needs behind an entered keyword query to 553  
make the incremental construction process faster. 554

## 5.10 Conclusion

In this chapter, we introduced QUICK, a system for guiding users in constructing 556  
semantic queries from keywords. QUICK allows users to query semantic data 557  
without any prior knowledge of its ontology. A user starts with an arbitrary keyword 558  
query and incrementally transforms it into the intended semantic query. In this way, 559  
QUICK integrates the ease of use of keyword search with the expressiveness of 560  
semantic queries. 561

The presented algorithms optimize this process such that the user can construct 562  
the intended query with near-minimal interactions. The greedy version of the 563  
algorithm exhibits a polynomial runtime behavior, ensuring its applicability on 564  
large-scale real-world scenarios. To our knowledge, QUICK is the first approach 565  
that allows users to incrementally express the intent of their keyword query. 566  
We presented the design of the complete QUICK system and demonstrated its 567  
effectiveness and practicality through an extensive experimental evaluation. 568

As shown in our study, QUICK can be further improved and extended in the 569  
following directions. (1) While QUICK currently works well on focused domain 570  
schemas, large ontologies pose additional challenges with respect to usability 571

as well as efficiency. To improve usability, we plan to make use of concept hierarchies to aggregate query construction options. In that way, we keep their number manageable and prevent the user from being overwhelmed by overly detailed options. To improve further on efficiency, we are working on an algorithm that streamlines the generation of the semantic query space. (2) A further field of improvement is making the user interface more intuitive, especially for users without expertise in semantic Web or databases. (3) Based on the improved user interface, we will conduct a user study to verify its suitability for nonexpert users and its effectiveness on a larger scale.

## References

1. Agrawal, S., Chaudhuri, S., Das, G.: DBXplorer: a system for keyword-based search over relational databases. ICDE (2002). DOI 10.1109/ICDE.2002.994693 582  
583
2. Bast, H., Weber, I.: The CompleteSearch engine: interactive, efficient, and towards IR & DB integration. CIDR (2007) 584  
585
3. Bernstein, A., Kaufmann, E.: Gino – a guided input natural language ontology editor. ISWC, pp. 144–157 (2006) 586  
587
4. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: a generic architecture for storing and querying RDF and RDF Schema. ISWC (2002) 588  
589
5. Broughton, V., Heather, L.: Classification schemes revisited: applications to web indexing and searching. *J. Inter. Catalog.* **2**(3/4), 143–155 (2000) 590  
591
6. Fang, L., Clement, T.Y., Weiyi, M., Abdur, C.: Effective keyword search in relational databases. Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 563–574. Chicago, Illinois, USA (2006) 592  
593  
594
7. Demidova, E., Zhou, X., Nejdl, W.: Iqp: incremental query construction, a probabilistic approach. ICDE 2010, pp. 349–352 (2010) 595  
596
8. Haller, H.: QuiKey – the smart semantic commandline (a concept). Poster and extended abstract presented at ESWC2008 (2008) 597  
598
9. Harth, A., Decker, S.: Optimized index structures for querying RDF from the Web. Proceedings of the 3rd Latin American Web Congress (2005) 599  
600
10. Harth, A., Umbrich, J., Hogan, A., Decker, S.: YARS2: a federated repository for querying graph structured data from the Web. ISWC/ASWC (2007) 601  
602
11. Jagadish, H.V., Chapman, A., Elkiss, A., Jayapandian, M., Li, Y., Nandi, A., Yu, C.: Making database systems usable. SIGMOD (2007) 603  
604
12. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations. Plenum Press, NY, USA (1972) 605  
606
13. Kaufmann, E., Bernstein, A.: How useful are natural language interfaces to the semantic web for casual end-users. ISWC/ASWC, pp. 281–294 (2007) 607  
608
14. Kaufmann, E., Bernstein, A., Zumstein, R.: Querix: a natural language interface to query ontologies based on clarification dialogs. ISWC, pp. 980–981 (2006) 609  
610
15. Kimelfeld, B., Sagiv, Y.: Finding and approximating top-k answers in keyword proximity search. PODS (2006). DOI <http://doi.acm.org/10.1145/1142351.1142377> 611  
612
16. Lei, Y., Uren, V.S., Motta, E.: Semsearch: a search engine for the semantic web. EKAW (2006) 613
17. Li, G., Ooi, B.C., Feng, J., Wang, J., Zhou, L.: Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. SIGMOD Conference (2008) 614  
615
18. Lopez, V., Uren, V., Motta, E., Pasin, M.: Aqualog: an ontology-driven question answering system for organizational semantic intranets. *J. Web Semant.* **5**(2), 72–105 (2007) 616  
617

19. Minack, E., Sauermann, L., Grimnes, G., Fluit, C., Broekstra, J.: The Sesame LuceneSail: RDF queries with full-text search. Tech. Rep. 2008-1, NEPOMUK (2008) 618  
619
20. Minack, E., Siberski, W., Nejdl, W.: Benchmarking fulltext search performance of RDF stores. Proceedings of the 6th European Semantic Web Conference (ESWC 2009), pp. 81–95. Heraklion, Greece (2009) 620  
621  
622
21. Möller, K., Ambrus, O., Josan, L., Handschuh, S.: A visual interface for building SPARQL queries in Konduit. International semantic web conference (posters & demos) (2008) 623  
624
22. Nandi, A., Jagadish, H.V.: Assisted querying using instant-response interfaces. SIGMOD (2007). DOI <http://doi.acm.org/10.1145/1247480.1247640> 625  
626
23. Neumann, T., Weikum, G.: RDF-3X: a RISC-style engine for RDF. Proc. VLDB Endowment 1(1), 647–659 (2008) 627  
628
24. Reichert, M., Linckels, S., Meinel, C., Engel, T.: Student's perception of a semantic search engine. IADIS CELDA, pp. 139–147. Porto, Portugal (2005) 629  
630
25. Ruckhaus, E., Vidal, M.E., Ruiz, E.: OnEQL: an ontology efficient query language engine for the semantic web. ALPSWS (2007) 631  
632
26. Russell, A., Smart, P.R.: NITELIGHT: a graphical editor for SPARQL queries. International semantic web conference (posters & demos) (2008) 633  
634
27. Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G.: Access path selection in a relational database management system. SIGMOD (1979) 635  
636
28. Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D.: Sparql basic graph pattern optimization using selectivity estimation. Proceedings of the 17th International Conference on World Wide Web, pp. 595–604. Beijing, China (2008) 637  
638  
639
29. Stojanovic, N., Stojanovic, L.: A logic-based approach for query refinement in ontology-based information retrieval systems. ICTAI 2004. 16th IEEE International Conference on Tools with Artificial Intelligence, pp. 450–457 (2004). DOI [10.1109/ICTAI.2004.13](http://doi.acm.org/10.1109/ICTAI.2004.13) 640  
641  
642
30. Stuckenschmidt, H., Vdovjak, R., Houben, G.J., Broekstra, J.: Index structures and algorithms for querying distributed RDF repositories. WWW, pp. 631–639 (2004) 643  
644
31. Tata, S., Lohman, G.M.: SQAK: doing more with keywords. SIGMOD (2008). DOI <http://doi.acm.org/10.1145/1376616.1376705> 645  
646
32. Tran, T., Cimiano, P., Rudolph, S., Studer, R.: Ontology-based interpretation of keywords for semantic search. ISWC (2007) 647  
648
33. Wang, H., Zhang, K., Liu, Q., Tran, T., Yu, Y.: Q2semantic: a lightweight keyword interface to semantic search. ESWC, pp. 584–598 (2008) 649  
650
34. Zenz, G., Zhou, X., Minack, E., Siberski, W., Nejdl, W.: From keywords to semantic queries – incremental query construction on the semantic web. J. Web Semat. 7(3), 166–176 (2009) 651  
652
35. Zhou, Q., Wang, C., Xiong, M., Wang, H., Yu, Y.: SPARK: adapting keyword query to semantic search. ISWC (2007) 653  
654

AUTHOR QUERIES

- AQ1. Kindly check the corresponding author.
- AQ2. In the sentence “As shown in Table 5.1,...” Please check if ‘rsp’ can be changed as ‘respectively’.
- AQ3. Please confirm the changes in the sentence “For example, [5] classifies...”.

UNCORRECTED PROOF

# Chapter 6

## Understanding the Semantics of Keyword Queries on Relational Data Without Accessing the Instance

Sonia Bergamaschi, Elton Domnori, Francesco Guerra, Silvia Rota,  
Raquel Trillo Lado, and Yannis Velegrakis

### 6.1 Introduction

The birth of the Web has brought an exponential growth to the amount of the information that is freely available to the Internet population, overloading users and entangling their efforts to satisfy their information needs. Web search engines such as Google, Yahoo, or Bing have become popular mainly due to the fact that they offer an easy-to-use query interface (i.e., based on keywords) and an effective and efficient query execution mechanism.

The majority of these search engines do not consider information stored on the *deep* or *hidden Web* [9, 28], despite the fact that the size of the deep Web is estimated to be much bigger than the surface Web [9, 47]. There have been a number of systems that record interactions with the deep Web sources or automatically submit queries to them (mainly through their Web form interfaces) in order to index their context. Unfortunately, this technique is only partially indexing the data instance. Moreover, it is not possible to take advantage of the query capabilities of data sources, for example, of the relational query features, because their interface is often restricted from the Web form. Besides, Web search engines focus on retrieving documents and

---

AQ1

S. Bergamaschi (✉) · E. Domnori · S. Rota  
DII-UNIMORE, via Vignolese 905 Modena, Italy  
e-mail: [sonia.bergamaschi@unimore.it](mailto:sonia.bergamaschi@unimore.it); [elton.domnori@unimore.it](mailto:elton.domnori@unimore.it); [silvia.rota@unimore.it](mailto:silvia.rota@unimore.it)

F. Guerra  
DEA-UNIMORE, v.le Berengario 51, Modena, Italy  
e-mail: [francesco.guerra@unimore.it](mailto:francesco.guerra@unimore.it)

R.T. Lado  
Informática e Ing. Sistemas, c/ María de Luna 50018 Zaragoza, Spain  
e-mail: [raqueltl@unizar.es](mailto:raqueltl@unizar.es)

Y. Velegrakis  
University of Trento, Trento, Italy  
e-mail: [vegias@disi.unitn.eu](mailto:vegias@disi.unitn.eu)

not on querying structured sources, so they are unable to access information based on concepts [31].

There are, on the other hand, numerous companies and organizations that have lots of data, so far in their own private (structured) data sources, and are willing to make them public on the Web. These data will remain unexploited unless they can be queried through keyword-based interfaces which is the de facto standard on the Web. There are already various studies on how to offer such a service [18, 41, 44, 48, 50], but they all require prior access to the instance in order to build an index that will allow them at runtime to understand what part of the database that a keyword in the query is referring to. This is not always feasible since companies and organizations may have their own restrictions and concerns about allowing an external service getting full access to the data.

A recent initiative that has attracted considerable attention is the notion of the *semantic Web*.<sup>1</sup> The idea of the semantic Web is to extend the data of the current Web with more structured data (typically expressed in RDF) that will allow them to be used by machines as well as by people. The success of the semantic Web depends on the existence of such kind of information and on the ability to query it. The semantic Web community has so far focused on ways to model this information. Allowing relational databases to be published on the semantic Web will provide the community with large amounts of data. However, it will also require the design and development of techniques and tools for querying that information in a way similar to the way it is currently done on the Web, i.e., through keyword queries. This means that any effort toward supporting keyword queries for structural data is of major importance for the specific community.

This chapter deals exactly with the problem of answering a keyword query over a relational database. To do so, one needs to understand the meaning of the keywords in the query, “guess” its possible semantics, and materialize them as SQL queries that can be executed directly on the relational database. The focus of the chapter is on techniques that do not require any prior access to the instance data, making them suitable for sources behind wrappers or Web interfaces or, in general, for sources that disallow prior access to their data in order to construct an index. The chapter describes two techniques that use semantic information and metadata from the sources, alongside the query itself, in order to achieve that. Apart from understanding the semantics of the keywords themselves, the techniques are also exploiting the order and the proximity of the keywords in the query to make a more educated guess. The first approach is based on an extension of the Hungarian algorithm [11] for identifying the data structures having the maximum likelihood to contain the user keywords. In the second approach, the problem of associating keywords into data structures of the relational source is modeled by means of a hidden Markov model, and the Viterbi algorithm is exploited for computing the mappings. Both techniques have been implemented in two systems called KEYMANTIC [5, 6] and KEYRY [7], respectively.

<sup>1</sup><http://www.w3.org/standards/semanticweb/>

The chapter is structured as follows. First, a motivating example is presented to introduce the reader to the problem and the challenges. Then the problem statement is formally formulated (Sect. 6.3). Sections 6.4 and 6.5 describe the two approaches in detail. Section 6.6 provides an overview of the related works and how they differ from the two approaches presented in this chapter.

## 6.2 Motivating Example

Let us assume that a virtual tourism district composed of a set of companies (travel agencies, hotels, local public administrations, tourism promotion agencies) wants to publish an integrated view of their tourism data about a location (see Fig. 6.1). Keymantic allows users to query that data source with a two-step process: firstly, the keyword query is analyzed for discovering its intended meaning then a ranked set of SQL queries, expressing the discovered meaning according to the database structure, is formulated.

Each keyword represents some piece of information that has been modeled in the database, but, depending on the design requirements of the data source, this piece might have been modeled as data or metadata. Thus, the first task is to discover what each keyword models in the specific data source and to which metadata/data may be associated to. The association between keywords and database needs to be approximate: the synonymous and polysemous terms might allow the discovery of multiple intended meanings, each one with a rank expressing its relevance. Let us consider, e.g., a query consisting of the keywords “Restaurant Naples.” For instance, a possible meaning of the query might be “find information about the restaurant called Naples.” In this case, the former keyword should be mapped into a metadata (the table Restaurant and the other one into a value of the attribute Name of the same table Restaurant). A user might have in mind other meanings for the same

Person				
Name	Phone	City	Email	
Saah	4631234	London	saah@aaa.bb	
Sevi	6987654	Auckland	eevi@bbb.cc	
Edihb	1937842	Santiago	edihb@ccc.dd	

Hotel				
id	Name	Address	Service	City
x123	Galaxy	25 Blicker	restaurant	Shanghai
cs34	Krystal	15 Tribeca	parking	Cancun
ee67	Hilton	5 West Ocean	air cond.	Long Beach

Restaurant				
id	Name	Address	Specialty	City
Rx1	Best Lobster	25, Margaritas	Seafood	Cancun
Rt1	Naples	200 Park Av.	Italian	New York

Reserved		
Person	Hotel	Date
Saah	x123	6/10/2009
Sevi	cs34	4/3/2009
Edihb	cs34	7/6/2009

City		
Name	Country	Description
Shanghai	China	...
Cancun	Mexico	...
Long Beach	USA	...
New York	USA	...

Fig. 6.1 A fraction of a database schema with its data

keywords, e.g., “find the restaurants that are located in the Naples Avenue,” or “in the Naples city,” or “that cook Naples specialties.” All these intended meanings give rise to different associations of the keyword Naples; attributes Address, City, or Specialty of the table Restaurant. This example shows also that keywords in a query are not independent: we expect that the keywords express different features of what the user is looking for. For this reason, we expect that in our example, “Naples” is a value referring to an element of the Restaurant table. If the user had formulated the keyword query “Restaurant name Naples,” the number of possible intended meanings would have been reduced, since the keywords name forces the mappings of Naples into the attribute Name of the table Restaurant. Notice that different intended meanings may generate a mapping from the same keyword both into metadata and into data values. For example, in our database, restaurant is the name of a table, but it is also one of the possible values of the attribute Service in the table Hotel. Finally, the order of the keywords in a query is also another element to be taken into account since related elements are usually close. If a user asks for “Person London restaurant New York,” one possible meaning of the query is that the user is looking for the restaurant in New York visited by people from London. Other permutations of the keywords in the query may generate other possible interpretations.

The second step in answering a keyword query concerns the formulation of an SQL query expressing one of the discovered intended meanings. In a database, semantic relationships between values are modeled either through the inclusion of different attributes under the same table or through join paths across different tables. Different join paths can lead to different interpretations. Consider, for instance, the keyword query “Person USA.” One logical mapping is to have the word Person corresponding to the table Person and the word USA to a value of the attribute Country of the table City. Even when this mapping is decided, there are different interpretations of the keywords based on the different join paths that exist between the tables Person and City. For instance, one can notice that a person and a city are related through a join path that goes through the City attribute referring to the attribute Name in the table City (determining which people in the database are from USA), through another path that is based on the table Hotel (determining which people reserved rooms of hotels in USA), and also through another path that is based on the table Restaurant (determining which people reserved a table in an American restaurant).

Finding the different semantic interpretations of a keyword query is a combinatorial problem which can be solved by an exhaustive enumeration of the different mappings to database structures and values. The large number of different interpretations can be brought down by using internal and external knowledge that helps in eliminating mappings that are not likely to lead to meanings intended by the user. For instance, if one of the provided keywords in a query is “320-463-1463,” it is very unlikely that this keyword refers to an attribute or table name. It most probably represents a value and, in particular, due to its format, a phone number. Similarly, the keyword “Bistro” in a query does not correspond to a table or an attribute in the specific database. Some auxiliary information, such as a thesaurus,

can provide the information that the word “bistro” is typically used to represent a restaurant; thus, the keyword can be associated to the Restaurant table.

135  
136

## 6.3 Problem Statement

137

**Definition 6.1.** A database  $D$  is a collection  $V_t$  of relational tables  $R_1, R_2, \dots, R_n$ .  
 Each table  $R$  is a collection of attributes  $A_1, A_2, \dots, A_{m_R}$ , and each attribute  $A$  has a domain, denoted as  $\text{dom}(A)$ . Let  $V_a = \{A \mid A \in R \wedge R \in V_t\}$  represent the set of all the attributes of all the tables in the database and  $V_d = \{d \mid d = \text{dom}(A) \wedge A \in V_a\}$  represent the set of all their respective domains. The *database vocabulary* of  $D$ , denoted as  $V_D$ , is the set  $V_D = V_t \cup V_a \cup V_d$ . Each element of the set  $V_D$  is referred to as a *database term*.

We distinguish two subsets of the database vocabulary: the *schema vocabulary*  $V_{SC} = V_t \cup V_a$  and the *domain vocabulary*  $V_{DO} = V_d$  that concerns the instance information. We also assume that a keyword query  $KQ$  is an ordered  $l$ -tuple of keywords  $(k_1, k_2, \dots, k_l)$ .

**Definition 6.2.** A configuration  $f_c(KQ)$  of a keyword query  $KQ$  on a database  $D$  is an injective function from the keywords in  $KQ$  to database terms in  $V_D$ . In other words, a configuration is a mapping that describes each keyword in the original query in terms of database terms.

The reason we consider a configuration to be an injective function is because we assume that (1) each keyword cannot have more than one meaning in the same configuration, i.e., it is mapped into only one database term; (2) two keywords cannot be mapped to the same database term in a configuration since overspecified queries are only a small fraction of the queries that are typically met in practice [22]; and (3) every keyword is relevant to the database content, i.e., keywords always have a correspondent database term. Furthermore, while modeling the keyword-to-database term mappings, we also assume that every keyword denotes an element of interest to the user, i.e., there are no stop words or unjustified keywords in a query. In this chapter, we do not address query cleaning issues. We assume that the keyword queries have already been preprocessed using well-known cleansing techniques.

Answering a keyword query over a database  $D$  means finding the SQL queries that describe its possible semantics in terms of the database vocabulary. Each such SQL query is referred to as an *interpretation* of the keyword query in database terms. An interpretation is based on a configuration and includes in its clauses all the database terms that are part of the image<sup>2</sup> of the query keywords through the configuration. In the current work, we consider only select-project-join (SPJ)

---

<sup>2</sup>Since a configuration is a function, we use the term image to refer to its output.

interpretations that are typically the queries of interest in similar works [2, 19], but interpretations involving aggregations [42] are part of our future work. 170  
171

**Definition 6.3.** An *interpretation* of a keyword query  $KQ = (k_1, k_2, \dots, k_l)$  on 172  
a database  $D$  using a configuration  $f_c^*(KQ)$  is an SQL query in the form select 173  
 $A_1, A_2, \dots, A_o$  from  $R_1 \text{ JOIN } R_2 \text{ JOIN } \dots \text{ JOIN } R_p$  where  $A'_1 = v_1 \text{ AND } A'_2 = v_2$  174  
AND  $\dots \text{ AND } A'_q = v_q$  such that the following holds: 175

- $\forall A \in \{A_1, A_2, \dots, A_o\} : \exists k \in KQ \text{ such that } f_c^*(k) = A$  176
- $\forall R \in \{R_1, R_2, \dots, R_p\} : (\text{i}) \exists k \in KQ : f_c^*(k) = R \text{ or (ii)} \exists k_i, k_j \in KQ : f_c^*(k_i) = R_i \wedge f_c^*(k_j) = R_j \wedge \text{exists a join path from } R_i \text{ to } R_j \text{ that involves } R$  177  
178
- $\forall "A' = v" \in \{A'_1 = v_1, A'_2 = v_2, \dots, A'_o = v_o\} : \exists k \in KQ \text{ such that } f_c^*(k) = \text{dom}(A') \wedge k = v$  179  
180
- $\forall k \in KQ : f_c^*(k) \in \{A_1, A_2, \dots, A_o, R_1, R_2, \dots, R_p, \text{dom}(A'_1), \dots, \text{dom}(A'_q)\}$  181

The existence of a database term in an interpretation is justified either by 182  
belonging to the image of the respective configuration or by participating in a join 183  
path connecting two database terms that belong to the image of the configuration. 184  
Note that even with this restriction, due to the multiple join paths in a database  $D$ , 185  
it is still possible to have multiple interpretations of a keyword query  $KQ$  given a 186  
certain configuration  $f_c^*(KQ)$ . We use the notation  $\mathcal{I}(KQ, f_c^*(KQ), D)$  to refer 187  
to the set of these interpretations and  $\mathcal{I}(KQ, D)$  for the union of all these sets for a 188  
query  $KQ$ . 189

Since each keyword in a query can be mapped into a table name, an attribute 190  
name, or an attribute domain, there are  $2 \sum_{i=1}^n |R_i| + n$  different mappings for each 191  
keyword, with  $|R_i|$  denoting the *arity* of the relation  $R_i$  and  $n$  the number of 192  
tables in the database. Based on this, and on the fact that no two keywords can 193  
be mapped to the same database term, for a query containing  $l$  keywords, there are 194  
 $\frac{|V_D|!}{(|V_D|-l)!}$  possible configurations. Of course, not all the interpretations generated by 195  
these configurations are equally *meaningful*. Some are more likely to represent the 196  
intended keyword query semantics. In the following sections, we will show how 197  
different kinds of metainformation and interdependencies between the mappings of 198  
keywords into database terms can be exploited in order to effectively and efficiently 199  
identify these meaningful interpretations and rank them higher. 200

## 6.4 The Hungarian Algorithm Approach

The generation of interpretations that most likely describe the intended semantics 202  
of a keyword query is based on semantically meaningful configurations, i.e., sets of 203  
mappings between each keyword and a database term. We introduce the notion of 204  
*weight* that offers a quantitative measure of the relativity of a keyword to a 205  
database term, i.e., the likelihood that the semantics of the database term are the 206  
intended semantics of the keyword in the query. The sum of the weights of the 207  
keyword–database term pairs can form a score serving as a quantitative measure of 208

the likelihood of the configuration to lead to an interpretation that accurately 209 describes the intended keyword query semantics. The range and full semantics of 210 the score cannot be fully specified in advance. They depend on the method used to 211 compute the similarity. This is not a problem as long as the same method is used 212 to compute the scores for all the keywords. This is the same approach followed 213 in schema matching [37] where a score is used to measure the likelihood that an 214 element of a schema corresponds to an element of another. 215

The naive approach for selecting the best configurations, and as a consequence, 216 generating the most prominent interpretations of a keyword query, is the computa- 217 tion of the score of each possible configuration and then selecting those that have 218 the highest scores. Of course, we would like to avoid an exhaustive enumeration of 219 all the possible configurations and compute only those that give high scores. The 220 problem of computing the mapping with the maximum score without an exhaustive 221 computation of the scores of all the possible mappings is known in the literature 222 as the problem of *bipartite weighted assignments* [13]. Unfortunately, solutions 223 to this problem suffer from two main limitations. First, apart from the mutual 224 exclusiveness, they do not consider any other interdependencies that may exist 225 between the mappings. Second, they typically provide only the best mapping instead 226 of a ranked list based on the scores. 227

To cope with the first limitation, we introduce two different kinds of weights: the 228 *intrinsic* and the *contextual* weights. Given a mapping of a keyword to a database 229 term, its intrinsic weight measures the likelihood that the semantics of the keyword 230 is that of the database term if considered in isolation from the mappings of all 231 the other keywords in the query. The computation of an intrinsic weight is based on 232 syntactic, semantic, and structural factors, such as attribute and relation names, or 233 other auxiliary external sources, such as vocabularies, ontologies, domains, common 234 syntactic patterns. On the other hand, a contextual weight is used to measure the 235 same likelihood but considering the mappings of the remaining query keywords. 236 This is motivated by the fact that the assignment of a keyword to a database term 237 may increase or decrease the likelihood that another keyword corresponds to a 238 certain database term. This is again based on observations that humans tend to write 239 queries in which related keywords are close to each other [22]. A similar idea has 240 already been exploited in the context of schema matching [30] with many interesting 241 results. To cope with the second limitation, we have developed a novel algorithm for 242 computing the best mappings. The algorithm is based on and extends the Hungarian 243 (a.k.a. Munkres) algorithm [11] and will be described in detail in Sect. 6.4.3. 244

A visual illustration of the individual steps in the keyword query translation task 245 is depicted in Fig. 6.2. A special data structure, called *weight matrix*, plays a central 246 role in these steps. The *weight matrix* is a two-dimensional array with a row for each 247 keyword in the keyword query and a column for each database term. The value of 248 a cell  $[i, j]$  represents the weight associated to the mapping between the keyword 249  $i$  and the database term  $j$ . Figure 6.3 provides an abstract illustration of a weight 250 matrix.  $R_i$  and  $A_j^{R_i}$  columns correspond to the relation  $R_i$  and the attribute  $A_j$  of 251  $R_i$ , respectively, while a column with an underlined attribute name  $\underline{A_j^{R_i}}$  represents 252

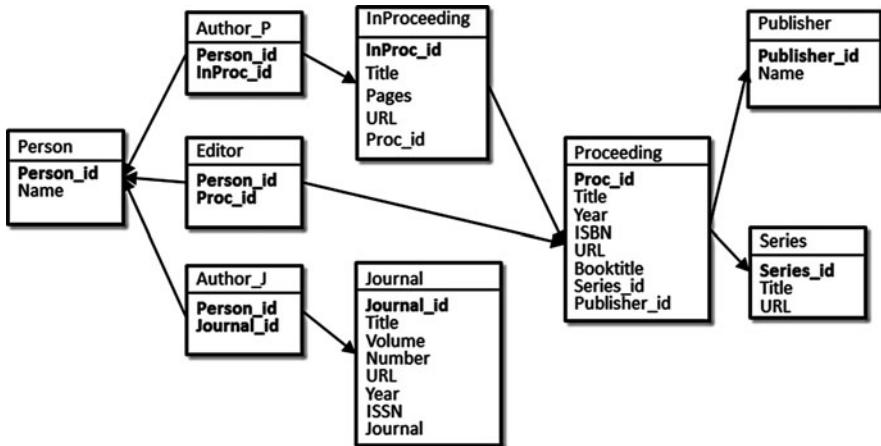


Fig. 6.2 Overview of the keyword query translation process

	$R_1$	$\dots$	$R_n$	$A_1^{R_1}$	$\dots$	$A_{n_1}^{R_1}$	$\dots$	$A_{n_n}^{R_1}$	$A_1^{R_1}$	$\dots$	$A_{n_1}^{R_1}$	$\dots$	$A_{n_n}^{R_1}$
$keyword_1$													
$keyword_2$													
$\dots$													
$keyword_k$													

Fig. 6.3 Weight table with its SW (light) and VW (dark) parts

the data values in the column  $A_j$  of table  $R_i$  may have, i.e., its domain. Two parts 253  
(i.e., submatrices) can be distinguished in the weight matrix. One corresponds to 254  
the database terms related to schema elements, i.e., relational tables and attributes, 255  
and the other one corresponds to attribute values, i.e., the domains of the attributes. 256  
We refer to database terms related to schema elements *schema database terms* and 257  
to those related to domains of the attributes *value database terms*. In Fig. 6.3, 258  
these two submatrices are illustrated with different shades of gray. We refer to the 259  
weights in the first submatrix *schema weights* and to those of the second *value 260  
weights*. We also use the notation *SW* and *VW* to refer either to the respective 261  
submatrix or to their values. The details of the individual steps of Fig. 6.2 are 262  
provided next. 263

*Intrinsic Weight Computation.* The first step of the process is the intrinsic weight 264  
computation. The output is the populated *SW* and *VW* submatrices. The compu- 265  
tation is achieved by the exploitation and combination of a number of similarity 266  
techniques based on structural and lexical knowledge extracted from the data source 267  
and on external knowledge, such ontologies, vocabularies, domain terminologies, 268  
etc. Note that the knowledge extracted from the data source is basically the 269  
meta-information that the source makes public, typically, the schema structure 270  
and constraints. In the absence of any other external information, a simple string 271  
272

comparison based on tree-edit distance can be used for populating the  $SW$  272 submatrix. For the  $VW$  submatrix, the notion of *semantic distance* [15] can always 273 be used in the absence of anything else. As it happens in similar situations [37], 274 measuring the success of such a task is not easy since there is no single correct 275 answer. In general, the more metainformation has been used, the better. However, 276 even in the case that the current step is skipped, the process can continue with 277 the weight matrix where all the intrinsic values have the same default value. The 278 computation of the intrinsic weights is detailed in Sect. 6.4.1. 279

*Selection of the Best Mappings to Schema Terms.* The intrinsic weights provide a 280 first indication of the similarities of the keywords to database terms. To generate 281 the prominent mappings, we need on top of that to take into consideration the 282 interdependencies between the mappings of the different keywords. We consider 283 first the prominent mappings of keywords to schema terms. For that, we work on 284 the  $SW$  submatrix. Based on the intrinsic weights, a series of mappings,  $M_1^S$ , 285  $M_2^S$ , ...,  $M_n^S$ , of keywords to schema terms are generated. The mappings are 286 those that achieve the highest overall score, i.e., the sum of the weights of the 287 individual keyword mappings. The mappings are partial, i.e., not all the keywords 288 are mapped to some schema term. Those that remain unmapped will play the role 289 of an actual data value and will be considered in a subsequent step for mapping 290 to value database terms. The selection of the keywords to remain unmapped is bed 291 on the weight matrix and some cutoff threshold. Those with a similarity below the 292 threshold remain unmapped. For each of the mappings  $M_i^S$ , the weights of its  $SW$  293 matrix are adjusted to take into consideration the context generated by the mapping 294 of the neighboring keywords. It is based on the observation that users form queries 295 in which keywords referring to the same or related concepts are adjacent [22, 45]. 296 The generation of the mappings and the adjustment of the weights in  $SW$  are 297 performed by our extension of the Hungarian algorithm that is described in detail 298 in Sect. 6.4.3. The output of such a step is an updated weight matrix  $SW_i$  and, 299 naturally, an updated score for each mapping  $M_i^S$ . Given the updated scores, some 300 mappings may be rejected. The selection is based on a threshold. There is no golden 301 value to set the threshold value. It depends on the expectations from the keyword 302 query answering systems. The higher its value, the less the interpretations generated 303 at the end, but with higher confidence. In contrast, the lower the threshold value, the 304 more the mappings with lower confidence. 305

*Contextualization of  $VW$  and Selection of the Best Mappings to Value Terms.* For 306 each partial mapping  $M_i^S$  of keywords to schema terms generated in the previous 307 step, the mappings of the remaining unmapped keywords to value terms need to 308 be decided. This is done in two phases. First, the intrinsic weights of the  $VW$  309 submatrix that were generated in Step 1 are updated to reflect the added value 310 provided by the mappings in  $M_i^S$  of some of the keywords to schema database terms. 311 This is called the process of contextualization of the  $VW$  submatrix. It is based on 312 the documented observation that users form queries in which keywords specifying 313 metadata information about a concept are adjacent or at let neighboring [22, 45]. 314 Thus, when a keyword is mapped to a schema term, it becomes more likely that an 315

adjacent keyword should be mapped to a value in the domain of that schema term. 316  
 The contextualization process increases the weights of the respective value terms to 317  
 reflect exactly that. For example, in the keyword query ‘‘Name Alexandria’’ 318  
 assume that the keyword **Alexandria** was found during the first step to be equally 319  
 likely the name of a person or of a city. If in Step 2 the keyword Name has been 320  
 mapped to the attribute Name of the table Person, the confidence that **Alexandria** 321  
 is actually the name of a person is increased; thus, the weight between that keyword 322  
 and the value database term representing the domain of attribute Name should be 323  
 increased, accordingly. In the second phase, given an updated  $VW_i$  submatrix, the 324  
 most prominent mappings of the remaining unmapped keywords to value database 325  
 terms are generated. The mappings are generated by using again the adapted 326  
 technique of the Hungarian algorithm (ref. Sect. 6.4.3). The result is a series of 327  
 partial mappings  $M_{ik}^V$ , with  $k = 1 \dots m_i$ , where  $i$  identifies the mapping  $M_i^S$  on 328  
 which the computation of the updated matrix  $VW_i$  is based. Given one such mapping 329  
<sup>AQ3</sup>  $M_{ik}^V$ , the value weights are further updated to reflect the mappings of the adjacent 330  
 keywords to value database terms, in a way similar to the one done in Step 2 for 331  
 the  $SW$  submatrix. The outcome modifies the total score of each mapping  $M_{ik}^V$ , and 332  
 based on that score, the mappings are ranked. 333

*Generation of the Configurations.* As a fourth step, each pair of a mapping  $M_{ik}^V$  334  
 together with its associated mapping  $M_i^S$  is a total mapping of the keywords to 335  
 database terms, forming a configuration  $C_{ik}$ . The score of the configuration is the 336  
 sum of the scores of the two mappings, or alternatively the sum of the weights in 337  
 the weight matrix of the elements  $[i, j]$  where  $i$  is a keyword and  $j$  is the database 338  
 term to which it is mapped through  $M_{ik}^V$  or  $M_i^S$ . 339

*Generation of the Interpretations.* Having computed the best configurations, the 340  
 interpretations of the keyword query, i.e., the SQL queries, can be generated. 341  
 The score of each such query is the score of the respective configuration. Recall, 342  
 however, that a configuration is simply a mapping of the keywords to database 343  
 terms. The presence of different join paths among these terms results in multiple 344  
 interpretations. Different strategies can be used to further rank the selections. 345  
 One popular option is the length of the join path [21], but other heuristics 346  
 found in the literature [19] can also be used. It is also possible that a same 347  
 interpretation be obtained with different configurations. A postprocessing analysis 348  
 and the application of data-fusion techniques [10] can be used to deal with this 349  
 issue. However, this is not the main focus of the current work, and we will not 350  
 elaborate further on it. We adopt a greedy approach that computes a query for 351  
 every alternative join path. In particular, we construct a graph in which each node 352  
 corresponds to a database term. An edge connects two terms if they are structurally 353  
 related, i.e., through a table-attribute-domain value relationship, or semantically, 354  
 i.e., through a referential integrity constraint. Given a configuration, we mark 355  
 all terms that are part of the range of the configuration “marked.” Then we run 356  
 a breath-first traversal (that favors shorter paths) to find paths that connect the 357  
 disconnected components of the graph (if possible). The final SQL query is then 358  
 constructed using the “marked” database terms and, in particular, the tables for 359

its from clause, the conditions modeled by the edges for its where clause, and the remaining attributes for its select clause. Then the process is repeated to find a different interpretation that will be based on a different join path. The final order of the generated interpretations is determined by the way the different paths are discovered and the cost of the configuration on which each interpretation is based.

It is important to note here that if the thresholds used in the above steps are all brought down to zero, then our technique is able to generate all the possible interpretations that can be defined on a database, even the most unlikely. In that sense, our technique is complete. The thresholds serve only to exclude from the results any interpretation that is not likely to represent the semantics of the keyword query, while the weights are used to provide the basis for a ranking metric.

### 6.4.1 *Intrinsic Weight Computation*

372

To compute the intrinsic weights, we need to compute the relevance between every query keyword and every database term. Some fundamental information that can be used toward this direction, and that is typically available, is the schema information. It may include the table and attribute names, the domains of the attributes, and very often, referential and integrity constraints, such as keys and foreign keys. Syntactic descriptions of the contents of an attribute (e.g., regular expressions) can also lead to a better matching of keywords to database terms since they offer indications on whether a keyword can serve as a value for an attribute or not. There are already many works that offer typical syntax for common attributes such as phone numbers, addresses, etc. [37], and have been used extensively and successfully in other areas. If access to the catalog tables is possible, assertion statements can offer an alternative source of syntactic information. In the same spirit, relevant values [8], i.e., clusters of the attribute domains, are also valuable auxiliary information. Furthermore, there is today a large volume of grammatical and semantic information that is publicly available on the Internet and can be used as a service. Examples include the popular WordNet and the many community-specific ontologies.

389

#### 6.4.1.1 *Weights for Schema Database Terms*

390

Finding matches between the flat list of keywords and the schema terms looks like the situation of schema matching [37] in which one of the schemas is the flat universal relation [29]. We follow a similar approach in which we employ a number of different similarity, measurement techniques and consider the one that offers the best result. One of these techniques is the string similarity [16]. For the string similarity, we further employ a number of different similarity metrics such as the Jaccard, the Hamming, the Levenshtein, etc., in order to cover a broad

397

**Algorithm 12:** Intrinsic SW matrix computation

---

**Data:**  $Q$ : Keyword Query,  $T$ : Schema Database Terms  
**Result:** SW matrix

```

begin
     $SW \leftarrow [0, 0, \dots, 0]$  ;
     $\Sigma \leftarrow \{ \text{Synonyms}(w,t), \text{Hyponyms}(w,t), \text{Hypernyms}(w,t), \text{StringSimilarity}(w,t) \dots \}$  ;
    for  $w \in Q$  do
        for  $e \in T$  do
             $sim \leftarrow 0$  ;
            for  $m \in \Sigma$  do
                if  $m(w, e) > sim$  then
                     $sim \leftarrow m(w, e)$  ;
                if  $ssim \leq threshold$  then
                     $sim \leftarrow 0$  ;
             $SW[w, c] = ssim * 100$  ;
    end

```

---

spectrum of situations that may occur. Since string similarity may fail in cases of 398 highly heterogeneous schemas that lack a common vocabulary, we also measure 399 the relativity of a keyword to schema database term based on their semantic 400 relationship. For that we employ public ontologies, such as SUMO,<sup>3</sup> or semantic 401 dictionaries such as WordNet, which can provide synonyms, hypernyms, hyponyms, 402 or other terms related to a given word. 403

Algorithm 12 describes the computation procedure of the intrinsic schema weight 404 matrix  $SW$ . The set  $\Sigma$  represents the similarity methods we employ. We have 405 a number of default methods that represent the state of the art in the area, but 406 additional methods can be included. Each such method takes as input two strings and 407 returns their respective similarity in a range between 0 and 1. We trust the method 408 that gives the highest similarity. If a similarity between a keyword and a schema 409 term is found below a specific threshold (that is set by the application), then the 410 similarity is set explicitly to 0. As a result, at the end of the procedure, there might 411 be rows in the matrix  $SW$  containing only zeros. These rows represent keywords 412 that are not similar enough to any of the schema database terms; thus, they will be 413 considered later as candidates for mapping to value database terms, i.e., domains of 414 the schema attributes. The fact that their rows in the  $SW$  matrix are 0 instead of 415 some low value makes the similarities of the keyword to the value database terms 416 that will be computed in a later step to be the dominating factor determining the 417 guess on the role a specific keyword can play. 418

*Example 6.1.* Consider the keyword query “people restaurant Naples” posed on the 419 database of Fig. 6.1. Figure 6.4 illustrates a fraction of the weight matrix containing 420 the intrinsic weights for the database terms derived from the tables Person and 421

AQ4

---

<sup>3</sup>[www.ontologyportal.org](http://www.ontologyportal.org)

	P	R	P.Na	P.Ph	P.Ci	P.Em	R.Id	R.Na	R.Ad	R.Sp	R.Ci	P.Na	P.Ph	P.Ci	P.Em	R.Id	R.Na	R.Ad	R.Sp	R.Ci
people	75	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1	1	1
restaurant	0	100	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1	1	1
Naples	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1	1	1

Fig. 6.4 Intrinsic weight SW (light gray) and VW (dark gray) matrix

Restaurant. Instead of the full names of tables and attributes, only the first letter of the tables and the first two letters of the attributes are used. The schema weights SW are the light gray-colored part of the matrix. Note that the keyword Naples has not been mapped to any of the schema terms since all the values of its row in SW are 0.

#### 6.4.1.2 Weights for Value Database Terms

427

For computing the intrinsic value weights, we mainly exploit domain information and base our decision on whether a keyword belongs to the domain of an attribute or not. Furthermore, we have adapted the notion of *semantic distance* [15] that is based on results retrieved by a search engine in order to evaluate the relatedness of two concepts. In particular, we define the *semantic relatedness*  $SR(x, y)$  of two terms  $x$  and  $y$  as

$$SR(x, y) = e^{-2NxD(x, y)}, \quad \text{where}$$

$$\begin{aligned} NxD(x, y) &= \{\max\{\log f(x), \log f(y)\} - \log f(x, y)\} / \\ &\quad \{\log N - \min\{\log f(x), \log f(y)\}\} \end{aligned}$$

with  $f(x)$  denoting the number of Web documents containing  $x$  and  $f(x, y)$  the number of documents containing both  $x$  and  $y$ , as these numbers are reported by specific search engines such as Google, Yahoo!, Cuil, Excite!, etc. The number  $N$  represents the number of documents indexed by the corresponding search engine. For our purpose, we compute the semantic relatedness of every keyword–attribute domain pair and this gives us an indication of the similarity degree between the keyword and the attribute domain. Information about possible values that an attribute can accept is also an important factor. The information is based on the explicit enumeration of values, as in the *relevant values* approach [8]. When a keyword is found among (or is similar to) the valid values that an attribute can get, the keyword receives a high weight. Additional comparison techniques include semantic measures based on external knowledge bases.

*Example 6.2.* For the keyword query introduced in Example 6.1, the intrinsic value weights are indicated in the *VW* part of Fig. 6.4, i.e., the dark gray-colored part. These weights have been computed by using domain knowledge and regular expressions. Note that these are the value weights; thus, the similarity is not between the keyword and the name of the attribute but concerns the compatibility of the keyword with the attribute domain.

451

### 6.4.2 Contextualization

452

The process of contextualization, as previously explained, exploits the interdependencies across mappings of different keywords. There are three different forms of contextualization that we consider. The first one increases the confidence of a keyword corresponding to an attribute (respectively, relation), if an adjacent keyword is mapped to the relation it belongs (respectively, one of the attributes of the relation). This is based on the generally observed behavior that users may sometimes provide more than one specification for a concept in a keyword query. For instance, they may use the keyword Person before the keyword Name to specify that they refer to the name of a person. The second form of contextualization is similar to the first, but applies on the value weights instead of the schema weights. The third and most important contextualization form is the one that updates the confidence of certain keywords corresponding to value database terms based on the mappings of other keywords to schema terms. The process consists of three phases and takes as input the value weight matrix  $VW$  and a partial mapping of keywords to schema database terms and returns an updated matrix  $VW$ . Let  $K$  be the ordered list of keywords in a query,  $M_i^S$  a partial mapping of keywords in  $K$  to schema terms, and  $K^S$  the subset of  $K$  containing the keywords for which  $M_i^S$  is defined, i.e., those that are mapped to some schema terms. We define the notion of *free trailing keywords* of a keyword  $k$ , denoted as  $T(k)$ , to be the maximum set  $k_1, k_2, \dots, k_m$  of consecutive keywords in  $K$  that are between two keywords  $k_s$  and  $k_e$  in the query and for which  $k_s = k$ ,  $M_i^S$  is defined for  $k_s$  and  $k_e$  and undefined for every  $k_i$  with  $i = 1 \dots m$ . The notion of *free preceding keywords* of a keyword  $k$ , denoted as  $P(k)$ , is defined accordingly, with  $k$  playing the role of  $k_e$ .

As an initialization step, all the weights in the rows of  $VW$  corresponding to keywords already mapped to database terms are set to zero. This is done to guarantee that in the three phases that are described next, none of these keywords will be mapped to a value term. In the first phase of the contextualization, for every keyword  $k$  mapped to a relation  $R$  through  $M_i^S$ , the weights of the trailing and preceding keywords  $T(k)$  and  $P(k)$  for terms representing the domains of the attributes of the relation  $R$  are increased by a constant value  $\Delta w$ . The rational of this action is that queries typically contain keywords that are generic descriptions for the values they provide. For instance, “Person Bill,” “Restaurant Naples,” etc., which means that consecutive keywords may correspond to a relation and a value of one of that relation’s attributes. During the second phase, for every keyword  $k$  mapped to an attribute  $A$  through  $M_i^S$ , the weights of the trailing and preceding keywords  $T(k)$  and  $P(k)$  with the database terms representing domains of attributes in the same relation as  $A$  are also increased by a constant value  $\Delta w$ . The rational of this rule is that consecutive keywords may represent value specifications and related values. An example is the query “Restaurant Vesuvio Pizza” intended to ask about the restaurant “Vesuvio” that has the “Pizza” as specialty. In the third phase, if a keyword  $k$  is mapped to an attribute  $A$ , the weights of the trailing and preceding keywords related to domains of attributes related to  $A$  through some

AQ5

join path are increased by the constant value  $\Delta w$ . The rational is that users use keywords referring to concepts that are semantically related, even if these concepts have been modeled in the database in different tables. An example of this situation is the keyword query “Phone Tribeca.” If phone is mapped to the attribute Phone of the relation Person, then the keyword Tribeca most likely represents the address of the department, which is stored in a separate table, and then the keyword query is about finding the phone number of the department that is on Tribeca. Note that the weight increase  $\Delta w$  can also be a percentage, instead of a constant, but our experimentations have shown no significant differences.

#### 6.4.3 Selecting the Best Mappings

Given a weight matrix, computing the best possible mapping of keywords to database terms is known as the *assignment problem* [13]. Unfortunately, the traditional solutions return the first best mapping while we would like them all in descending order, or at least the top- $k$ . Furthermore, we need a solution that, during the computation of the best mappings, takes into consideration interdependencies of the different assignments, i.e., the contextualization process. For this reason, we have adapted the popular systematic Hungarian, a.k.a. Munkres, algorithm [11] in order not to stop after the generation of the best mapping but to continue to the generation of the second best, the third, etc. Furthermore, some of its internal steps have been modified so that the weight matrix is dynamically updated every time that a mapping of a keyword to a database term is decided during the computation.

The execution of the algorithm consists of a series of iterative steps that generate a mapping with a maximum score. Once done, the weight matrix is modified accordingly to exclude the mapping that was generated and the process continues to compute the mapping with the second largest score, etc. More specifically, the maximum weight of each row is first identified and characterized as *maximum*. If the characterized-as-maximum weights are all located in different columns, then a mapping is generated by associating for each of the characterized-as-maximum weights the keyword and the database term that correspond to its respective row and column. On the other hand, if there is a column containing more than one weight characterized as maximum, all maximums in the column except the one with the maximum value lose their characterization as maximum. This last action means that some of the rows are left without some characterized weight. The values of the weights in these rows are then updated according to a number of contextual rules mentioned in Sect. 6.4. This is the effect of the mapped keywords to those that have remained unmapped.

In the sequel, for each row with no characterized weight, the one with the maximum value that belongs to a column corresponding to a database term different from the previous one is selected and characterized as *maximum*. If this leads to a matrix that has each characterized weight in a different column, then a mapping

**Algorithm 13:** Keyword to db term mapping selection

**Data:**  $I(i_{ij})$  where  $I \equiv SW$  or  $I \equiv VW$   
**Result:**  $M^I = \{M_1^I, \dots, M_z^I\}$ : Mappings generated by  $I$

```

MAPPING( $I, W_{MAX}$ );
begin
     $tempM = \bigcup i_{pt} \leftarrow HUNGARIAN_{Ext} * (I);$ 
     $W \leftarrow \sum i_{pt};$ 
     $M^I \leftarrow tempM;$ 
    if ( $W > c * W_{MAX}$ ) then
         $W_{MAX} \leftarrow W;$ 
    while ( $W > c * W_{MAX}$ ) do
        for  $i_{pt} \in tempM$  do
             $i_{pt} \in I \leftarrow -100;$ 
            Mapping( $I, W_{MAX}$ );
    end

```

is generated as above or the process of uncharacterizing some of the values as 535  
previously described is repeated. 536

Once a mapping of all the keywords has been formed, it is included in the set 537  
of mappings that will be returned by the algorithm. Then, the algorithm needs 538  
to be repeated to generate additional mappings. To avoid recomputing the same 539  
assignments, the algorithm is reexecuted cyclically with a new matrix as input. The 540  
new matrix is the old one modified to exclude mappings that have already been 541  
considered in previous runs of the algorithm. This is done by setting the values of 542  
the respective weights to a large negative number, forcing the algorithm to never 543  
select them again. This whole process is repeated until the scores of the mappings 544  
that the algorithm generates fall below some specific threshold. By construction, the 545  
most prominent mapping is the one that is first reported by this task. 546

The original Hungarian algorithm for rectangular matrices has an  $O(n^2 * m)$  547  
complexity [11], where  $n$  is the number of keywords and  $m$  is the number of 548  
database terms. Extending the algorithm to consider dynamic weights as described 549  
above brings the complexity to  $O(n^3 * m^2)$  which is due to the fact that a mapping 550  
may affect any other mapping; thus, in the worst case,  $(n - 1) * (m - 1)$  weight 551  
updates may take place. Nevertheless, this worst case rarely happens since only a 552  
subset of the matrix is updated in each iteration and, due to the threshold, not all the 553  
possible updates are evaluated. 554

Algorithm 13 depicts the overall process of computing the set of most prominent 555  
mappings of a set of keywords to database terms, given a weight matrix. The 556  
expression  $HUNGARIAN_{ext}$  refers to our extended version of the Hungarian 557  
algorithm. 558

*Example 6.3.* Figure 6.5 illustrates a VW matrix similar to the one of Fig. 6.4, 559  
but with additional keywords to better demonstrate the steps described in this 560  
section. The initial version of the matrix is the one composed of the first five 561  
lines. The lines of the keywords people and restaurant will remain unchanged since 562

	P.Na	P.Ph	P.Ci	P.Em	R.Id	R.Na	R.Ad	R.Sp	R.Ci
<i>people</i>	0	0	0	0	0	0	0	0	0
<i>restaurant</i>	0	0	0	0	0	0	0	0	0
<i>Vesuvio</i>	50	0	50	0	0	75	50	50	50
<i>pizza</i>	35	0	10	0	0	32	40	55	45
<i>Naples</i>	30	0	40	0	0	20	25	50	45
<i>Vesuvio</i>	50	0	50	0	0	75	50	50	50
<i>pizza</i>	35	0	10	0	0	32	40	55	45
<i>Naples</i>	30	0	40	0	0	20	25	50	45
<i>Vesuvio</i>	45	0	45	0	0	75	55	55	55
<i>pizza</i>	30	0	5	0	0	37	45	55	50
<i>Naples</i>	30	0	40	0	0	20	25	55	50
<i>Vesuvio</i>	50	0	50	0	0	75	50	50	50
<i>pizza</i>	35	0	10	0	0	32	40	55	45
<i>Naples</i>	30	0	40	0	0	20	25	50	45

**Fig. 6.5** Weight matrix during best mapping computation

the keywords are mapped to schema terms, and for this reason, they are omitted from the subsequent versions. The weights in white cells are those characterized as maximum. Each one is the largest weight in its row. Note that for column R.Sp, there are more than one weight characterized as maximum. From those, only the largest one is kept, in our case, 55. This leaves the row of keyword *Naples* with no weight characterized as maximum. The result is the second VW matrix illustrated in Fig. 6.5. The three characterized weights suggest a mapping for the keywords *Vesuvio* and *pizza*. Given these mappings, the weights are adjusted to reflect the interdependences according to the contextual rules. For instance, the mapping of *Vesuvio* to the database term R.Na triggers an increase in the weights of the database terms on the attributes in the same table. The result of firing the contextual rules is the third matrix in Fig. 6.5. In the updated matrix, the highest value is 49, which is in the column R.Sp, but cannot be chosen since the keyword *pizza* is already mapped to it. The second largest weight, 45, is in a column of a database term that no keyword is mapped to, and it is becoming characterized as maximum. The final outcome is the fourth matrix of Fig. 6.5 and, based on this, the mappings of keywords *Vesuvio*, *pizza*, and *Naples* to the database terms R.Na, R.Sp, and R.Ci, respectively, which is added to the mapping results generated by the algorithm. After that, the mapping is again considered for generating a new input for the algorithm. Four new matrices are derived from it, each one having one of the weights that was in a white cell in the last matrix reduced. For each obtained matrix, the whole process starts again from the beginning.

## 6.5 The Viterbi Approach

586

In an effort to define the configuration function, we can divide the problem of matching a whole query to database terms into smaller subtasks. In each subtask the best match between a single keyword and a database term is found. Then the final solution to the global problem is the union of the matches found in the subtasks. This approach works well when the keywords in a query are independent of each other, meaning that they do not influence the match of the other keywords to database terms. Unfortunately, this assumption does not hold in real cases. On the contrary, interdependencies among keyword meanings are of fundamental importance in disambiguating the keyword semantics. 587 588 589 590 591 592 593 594 595

In order to take into account these interdependencies, we model the matching function as a sequential process where the order is determined by the keyword ordering in the query. In each step of the process, a single keyword is matched against a database term, taking into account the result of the previous keyword match in the sequence. This process has a finite number of steps, equal to the query length, and is stochastic since the matching between a keyword and a database term is not deterministic: the same keyword can have different meanings in different queries and hence being matched with different database terms; vice versa, different database terms may match the same keyword in different queries. This type of process can be modeled, effectively, by using a hidden Markov model (HMM for short), which is a stochastic finite state machine where the states are hidden variables. 596 597 598 599 600 601 602 603 604 605 606 607

An HMM models a stochastic process that is not observable directly (it is hidden), but it can be observed indirectly through the observable symbols produced by another stochastic process. The model is composed of a finite number  $N$  of states. Assuming a time-discrete model, at each time step a new state is entered based on a *transition probability distribution*, and an observation is produced according to an *emission probability distribution* that depends on the current state, where both these distributions are time independent. Moreover, the process starts from an initial state based on an *initial state probability distribution*. We will consider first-order HMMs with discrete observations. In these models, the *Markov property* is respected, i.e., the transition probability distribution of the states at time  $t + 1$  depends only on the current state at time  $t$  and it does not depend on the past states at time  $1, 2, \dots, t - 1$ . Moreover, the observations are discrete: there exists a finite number,  $M$ , of observable symbols; hence, the emission probability distributions can be effectively represented using *multinomial distributions* dependent on the states. More formally, the model consists of the following: (1) a set of states  $S = \{s_i\}$ ,  $1 \leq i \leq N$ ; (2) a set of observation symbols  $V = \{v_j\}$ ,  $1 \leq j \leq M$ ; (3) a transition probability distribution  $A = \{a_{ij}\}$ ,  $1 \leq i \leq N$ ,  $1 \leq j \leq N$  where 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625

$$a_{ij} = P(q_{t+1} = s_j | q_t = s_i) \quad \text{and} \quad \sum_{0 < j < N} a_{ij} = 1$$

(4) an emission probability distribution  $B = \{b_i(m)\}$ ,  $1 \leq i \leq N$ ,  $1 \leq m \leq M$  626  
where 627

$$b_i(m) = P(o_t = v_m | q_t = s_i) \quad \text{and} \quad \sum_{0 < m < M} b_i(m) = 1$$

and (5) an initial state probability distribution  $\Pi = \{\pi_i\}$ ,  $1 \leq i \leq N$  628  
where 629

$$\pi_i = P(q_1 = s_i) \quad \text{and} \quad \sum_{0 < i < N} \pi_i = 1$$

Based on the above, the notation  $\lambda = (A, B, \Pi)$  will be used to indicate an 630  
HMM. In our context, the keywords inserted by the user are the observable part of 631  
the process, while the correspondent database terms are the unknown variables that 632  
have to be inferred. For this reason, we model the keywords as observations and 633  
each term in the database vocabulary as a state. 634

### 6.5.1 Setting HMM Parameters

635

In order to define an HMM, its parameters have to be identified. This is usually done 636  
using a training algorithm that, after many iterations, converges to a good solution 637  
for the parameter values. In our approach, we introduce some heuristic rules that 638  
allow the definition of the parameter values even when no training data are available. 639  
The HMM parameter values are set by exploiting the semantics collected from the 640  
data source metadata. In particular: 641

*The transition probabilities* are computed using heuristic rules that take into account 642  
the semantic relationships that exist between the database terms (aggregation, 643  
generalization, and inclusion relationships). The goal of the rules is to foster the 644  
transition between database terms belonging to the same table and belonging to 645  
tables connected through foreign keys. The transition probability values decrease 646  
with the distance of the states, e.g., transitions between terms in the same table 647  
have higher probability than transitions between terms in tables directly connected 648  
through foreign keys, which, in turn, have higher probability than transitions 649  
between terms in tables connected through a third table. 650

*The emission probabilities* are computed on the basis of similarity measures. In par- 651  
ticular, two different techniques are adopted for the database terms in  $V_{SC}$  and  $V_{DO}$ . 652  
We use the well-known *edit distance* for computing the lexical similarity between 653  
the keywords and each term (and its synonyms extracted from WordNet<sup>4</sup>) in the 654

---

<sup>4</sup><http://wordnet.princeton.edu>

schema vocabulary  $V_{SC}$ . On the other side, the similarity between keywords and the terms in the domain vocabulary  $V_{DO}$  is based on domain compatibilities and regular expressions. We use the calculated similarity as an estimate for the conditional probability  $P(q_t = s_i | o_t = v_m)$ , then, using the Bayes theorem, we calculate the emission probability  $P(o_t = v_m | q_t = s_i)$ . Note that the model is independent of the similarity measure adopted. Other more complex measures that take into account external knowledge sources (i.e., public ontologies and thesauri) can be applied without modifying the model.

The initial state probabilities are estimated by means of the scores provided by the HITS algorithm [25]. The HITS algorithm is a link analysis algorithm that calculates two different scores for each page: *authority* and *hub*. A high authority score indicates that the page contains valuable information with respect to the user query, while a high hub score suggests that the page contains useful links toward authoritative pages. This algorithm has been adapted to our context in order to rank the tables in a database based on their authority scores. The higher the rank, the more valuable is the information stored in the tables. For this reason, the authority score is used as an estimate of the initial state probabilities.

### 6.5.1.1 Adapted HITS Algorithm

In order to employ the HITS algorithm, we build the database graph  $G_D = (V_t, E_{fk})$  which is a directed graph where the nodes are the database tables in  $V_t$  and the edges are connections between tables through foreign keys, i.e., given two tables  $R_i, R_j$ , there exists an edge in  $E_{fk}$  from  $R_i$  to  $R_j$ , denoted as  $R_i \rightarrow R_j$ , if an attribute in  $R_j$  is referred by a foreign key defined in  $R_i$ . Let  $A$  be the  $n \times n$  adjacency matrix of the graph  $G_D$

$$A = [a_{ij}], a_{ij} = 1 \text{ iff } e_{ij} \in E_{fk}, a_{ij} = 0 \text{ iff } e_{ij} \notin E_{fk}$$

In our approach, we use the modified matrix  $B$  that takes into account the number of attributes minus the number of foreign keys in a table (foreign keys are considered as links):

$$B = [b_{ij}], b_{ij} = a_{ij} \cdot (|R_i| - \|\{R_i \rightarrow R_j, 1 \leq j \leq n\}\|)$$

Let us define the authority weight vector  $auth$  and the hub weight vector  $hub$

$$auth^T = [u_1, u_2, \dots, u_n] \text{ and } hub^T = [v_1, v_2, \dots, v_n]$$

The algorithm initializes these vectors with uniform values, e.g.,  $\frac{1}{n}$ , then the vectors are updated in successive iterations as follows:

$$\begin{cases} auth = B^T \cdot hub \\ hub = B \cdot auth \end{cases}$$

At each iteration, a normalization step is performed to obtain authority and hub scores in the range [0, 1]. After few iterations, the algorithm converges and the authority scores are used as estimate for the initial state probabilities.

*Example 6.4.* Let us consider the database in Fig. 6.1. This database generates states, one for each database term. Concerning the transition probability distribution, the heuristic rules foster transitions between database terms of the same table or in tables connected via foreign key. According to this, the most probable states subsequent to the state associated to the table *People* are the states associated to the names and the domains of the attributes *Name*, *Phone*, *City*, etc., then, the state associated to the tables *Booked*, *Reserved*, and *City*, subsequently, the states associated to the tables *Hotel* and *Restaurant*. We use different similarity measures for computing the emission probability distribution. Domain compatibility and regular expressions are used for measuring the probabilities of the values associated to states of terms in the  $V_{DO}$ . The probabilities of values associated to states if terms in the  $V_{SC}$  are computed on the basis of lexical similarity computed on the term and on its synonyms, hypernyms, and hyponyms. For example, we consider “Brasserie” as synonym of restaurant.

AQ6

### 6.5.2 Decoding the HMM

702

By applying the Viterbi algorithm to an HMM, we find the state sequence which has the highest probability of generating the observation sequence. If we consider the user keyword query as an observation sequence, the algorithm retrieves the states (i.e., the data source elements) that more likely represent the intended meaning of the user query. Note that, in general, the  $N$  best sequences, each one terminating in a different state, found by the Viterbi algorithm during the recursion are not the top- $N$  best sequences. The Viterbi algorithm divides the solution space (the number of configurations between  $l$  keywords and  $d$  database terms is  $P(l, d) = \frac{l!}{(l-d)!}$ ) into  $N$  subspaces, where each subspace contains all the sequences that end in state  $S_i$ ,  $1 \leq i \leq N$ . The algorithm finds the best solution in each subspace, then orders them to find the best overall solution. This process guarantees to find only the single best solution.

The Viterbi algorithm is not a good choice for facing the general problem of finding the top- $K$  solutions. The algorithm finds all  $K$  solutions only in the particular case where  $K \leq N$  and each subspace contains 0 or 1 of the top- $K$  solutions. In the worst scenario, when all the top- $K$  solutions are concentrated in a single subspace, the algorithm finds just the single best one. Moreover, we do not know in advance how the  $K$  solutions are distributed in the  $N$  subspaces. In order to

solve the general problem of finding the overall best  $K$  sequences, we implemented 721  
 a generalization of the Viterbi algorithm, called f, which keeps track of the best  $K$  722  
 solutions in each subspace and guarantees to find the top- $K$  sequences even in the 723  
 worst scenario. 724

Our implementation of List Viterbi is presented in pseudocode in Algorithm 14. 725  
 The algorithm is a generalization of the Viterbi algorithm because for  $K = 1$ , 726  
 it finds the same solution as the original algorithm, and for  $K = \frac{l!}{(l-d)!}$ , it finds 727  
 all the existing solutions and orders them globally. The algorithm is based on 728  
 a tree structure that memorizes the sequences found (each sequence starts from 729  
 the root and ends in one of the leaves). Each node at distance  $t$  from the root 730  
 represents a state  $q_t$  traversed in a particular sequence at step  $t$ . The tree structure 731  
 is built iteratively by the List Viterbi algorithm. In the first step, the tree height is 732  
 1 and the leaves represent all possible initial states given the observation sequence 733  
 $O = (o_1, o_2, \dots, o_T)$ , i.e., there are at maximum  $N$  leaves at this step. In all other 734

---

**Algorithm 14:** List Viterbi
 

---

**Data:**  $\lambda = (A, B, \Pi) : \text{HMM}$ ,  $KQ = (k_1, k_2, \dots, k_l) : \text{a keyword query}$ ,  $K : \text{top-K}$   
**Result:**  $Q[K][l] : \text{top-K state sequences}$ ,  $P[K][l] : \text{top-K state sequences probabilities}$

```

Compute_Tree( $A, B, \Pi, KQ, k$ );
begin
  Tree  $\leftarrow$  newnode(root);
  for state  $S_i$  do
     $\delta_0 \leftarrow b_i \cdot \pi_i$ ;
    appendNode(( $S_i, \delta_0$ ), Tree);
  L[]  $\leftarrow$  leaves(Tree);
  for  $k_i \in KQ$  do
    array Tmp[N][K];
    for state  $S_i$  do
      for  $j \in L$  do
         $\delta_k \leftarrow a_{ij} \cdot b_i \cdot \delta_{k-1}$ ;
        appendNode(( $S_i, \delta_k$ ), Tmp[ $S_i$ ][ ]);
        insert(Tmp[ $S_i$ ][ ], Tree);
      L[]  $\leftarrow$  L[] + Tmp[ $S_i$ ][ ];
      orderLeafs(L[]);
    end
  Backtracking(Tree);
  begin
    t  $\leftarrow l$ ;
    for  $j \in L$  do
      Q[j][t]  $\leftarrow$  state(L[j]);
      P[j][t]  $\leftarrow$  probability(L[j]);
      for  $t = l - 1, l - 2, 1$  do
        Q[j][t]  $\leftarrow$  state(parent(Q[j][t + 1], Tree));
        P[j][t]  $\leftarrow$  probability(parent(Q[j][t + 1], Tree));
  end

```

---

steps  $t$ , where  $2 \leq t \leq T$ , a maximum of  $K$  leaves representing a state  $q_t$  are 735 added to the tree, i.e., the tree memorizes up to  $K$  best paths that end in state  $q_t$ . 736 A maximum of  $K \cdot N$  leaves are added to the tree at each step. After the tree is built, 737 the  $K \cdot N$  leaves are ordered based on their probability and the first  $K$  are the top- $K$  738 solutions we were looking for. The complete sequences are then extracted from the 739 tree by backtracking the paths from the leaves to the root. 740

*Example 6.5.* The top-3 results of the keyword query “people restaurant 741 Naples” are the ones mapping the keyword people into the table *People*, 742 restaurant into the table *Restaurant*, and Naples into the domains of the 743 attributes *Name*, *Specialty*, and *City*, respectively. Both the solutions have the same 744 transition probabilities but different emission probabilities since the likelihood of the 745 mapping of Naples into the attribute *City* is higher than the one into *Restaurant* 746 and *Specialty*. 747

## 6.6 Related Work

748

The popularity of keyword searching is constantly increasing. For more than 749 a decade, it has been the successful model in IR for text databases [40] and 750 the Web [12]. It has also been adopted by the data management community in 751 the context of XML [17, 26, 43]. To answer a keyword query over XML data, the 752 appearances of the query keywords are first identified within the documents (pos- 753 sibly through some free-text search) and then combined together into meaningful 754 lowest common ancestor structures (MLCAS). A score is computed based on this 755 structure, and according to this score, the respective XML documents containing 756 the MLCAS are ranked and returned to the user. XML benefits from the fact 757 that its basic information model is the “document,” which is the same as in IR; 758 thus, many IR techniques can be easily employed in the XML context. On the 759 other hand, keyword search in relational databases is particularly challenging [42], 760 first because the database instances are way larger and second because the basic 761 model is fundamentally different, making hard the identification of the information 762 units that are to be returned. Nevertheless, keyword search over relational data 763 is particularly appealing, and there are already many interesting proposals in the 764 scientific literature [14, 49]. DISCOVER [19] and DBXplorer [2] have been among 765 the first such systems. The typical approach is to build a special index on the contents 766 of the database and then to use that index to identify the appearances of the query 767 keywords in the attribute values. Many approaches use inverted indexes [1, 19, 39] 768 for that purpose, while others, like DBXplore, use symbol tables. A symbol table is 769 an index consisting of a set of triples  $\langle value, attribute, relation \rangle$ , used to map every 770 value to its schema information. Once the keywords are located, the different ways 771 by which their respective tuples are connected are discovered, forming the so-called 772 “joining network” of tuples or tuple trees that often become the information unit 773 returned to the user. The joining networks are constructed either directly from the 774 instances or by building query expressions and evaluating them [19]. DISCOVER 775

is interested in finding total and minimal joining networks. A joining network is 776 total if each keyword query is contained in at least one tuple of the network, and 777 minimal if the removal of any tuple makes the network no longer total. More 778 recent approaches [36] are oriented toward reducing the number of tuples that need 779 to be considered in order to improve previous techniques. BANKS [1] follows a 780 similar approach but employs the Steiner tree algorithm to discover how the tuples 781 are associated. SQAK [42] is another system that is based on the same generic 782 principles but focuses on the discovery of aggregate SQL expressions that describe 783 the intended keyword query semantics. Since the set of possible answers may be 784 large, and since the results are already ranked, the above systems typically return 785 the top- $k$  results. 786

As have happened in all the above approaches, the two methods that we 787 presented in this chapter are also trying to identify the possible semantics (i.e., 788 expected answers) to the ambiguous keywords queries and rank the results. The 789 fundamental difference is that they do not assume any a priori access to the 790 database instance. Unavoidably, the approaches are based on schema and metadata, 791 i.e., the intensional information, which makes them applicable to scenarios where 792 the other techniques cannot work. Nevertheless, presented approaches should not 793 be seen as an alternative to the above methods. Since they operate on different 794 information, i.e., the metadata, they can be used to enhance these techniques 795 by providing a better exploitation of the metainformation and the relationships 796 among the keywords. The idea of using schema information and keyword semantics 797 has been considered in one of the approaches [27], but this is only limited to 798 word semantics based on WordNet. The presented two approaches go further by 799 combining not only additional semantic similarity techniques, similar to those used 800 in schema matching [37], but also on syntactic and structural information. Data 801 behavior has also been considered [42]. All these works are complementary. 802

Another distinction is on the relationship among the keywords. Existing 803 approaches compute the relationship among the keywords by considering the 804 relationships of the data values in the database instance. It is believed by many 805 that the keyword query should be the one driving the translation and not the data. 806 The presented approaches take into consideration in a systematic way the position 807 of the keywords in the query itself and the interdependencies between the matchings 808 of the keywords to the database structure. Furthermore, they map the keywords to 809 schema elements that will form the SQL query, instead of mapping them to the 810 tuples themselves [42]. This allows them, without performing any query evaluation, 811 to present these queries to the user and communicate the considered semantics. Of 812 course, this is an optional step, but in highly heterogeneous environments, this is 813 the norm for matching and mapping techniques [33]. 814

A related field is keyword disambiguation where several approaches have been 815 developed. For example, in [4] an incremental technique based on WordNet and 816 description logics is proposed, in [46] context and ontologies are exploited for 817 removing ambiguity, and in [34] attribute disambiguation enables faceted searches. 818

With specific reference to the KEYRY approach, a large number of techniques 819 based on HMM that have been proposed typically address the following three basic 820

problems [3]: (1) *finding the MLSS*, i.e., given an HMM model  $\lambda$  and a sequence of observations  $O_l$ , find the correspondent state sequence  $Q_l$  that has the highest probability of generating  $O_l$ ; (2) *evaluation problem*, i.e., given a model  $\lambda$  and a sequence of observations  $O_l$ , evaluate the probability that the model  $\lambda$  has generated the observations  $O_l$ ; and (3) *learning the parameters*, i.e., given a training dataset of observation sequences  $O$ , learn the model  $\lambda$  that maximizes the probability of generating  $O$ . This chapter described a way to apply the List Viterbi algorithm for addressing the first problem, i.e., decoding the HMM by computing the top-K answers to a keyword query. Other interesting applications of List Viterbi have been proposed especially in the data transmission field, where it has been exploited for detecting the correct message in cases of transmission errors [38] and for source-channel coding of images [20].

In [23], the Baum-Welch, the Viterbi training, and the Monte Carlo EM training are described, and an extension of the latter two approaches is proposed in order to make them more efficient. Since these algorithms can in practice be slow and computationally expensive, several techniques have been proposed for improving the learning under particular conditions: Lember and Koloydenko [24] propose an “adjusted Viterbi training” with the same complexity level as the Viterbi training algorithm, but computing more accurate results. Differently from these proposals, the presented scenario requires an online training approach in order to take into account the user’s feedback as soon as it is received.

Finally, the presented methods can also find applications in the field of graphical tools that assist the user in formulating queries [32]. By finding the different interpretations of a keyword query, one could detect related schema structures, make suggestions, and guide the user in the query formulation. Furthermore, in cases of exploratory searches, the user can use the generated interpretations as a way to explore an (unknown) data source and understand better its semantics.

## 6.7 Conclusion

848

The chapter presented an overview of two methods for translating keyword queries over a relational database into SQL so that they can be executed. The main focus was in a situation in which the contents of the database are unknown and the only information available is the metainformation, i.e., schema and constraints. In both cases, the idea was to understand the semantics of the keywords in order to correctly map them to database structures and then use this mapping to guide the query generation process. The first method was an extension of the Hungarian algorithm and the second of the Viterbi algorithms. Preliminary experiments have shown that both approaches work equally well; however, a clear understanding on when each one performs better is still not clear.

849  
850  
851  
852  
853  
854  
855  
856  
857  
858

## References

1. Aditya, B., Bhalotia, G., Chakrabarti, S., Hulgeri, A., Nakhe, C., Parag, Sudarshan, S.: Banks: browsing and keyword searching in relational databases. VLDB, pp. 1083–1086. Morgan Kaufmann, New York (2002) 860  
861  
862
2. Agrawal, S., Chaudhuri, S., Das, G.: Dbxplorer: a system for keyword-based search over relational databases. ICDE, pp. 5–16. IEEE Computer Society, Silver Spring, MD (2002) 863  
864
3. Alpaydin, E.: Introduction to Machine Learning, 2nd edn. MIT, Cambridge, MA (2010) 865
4. Bergamaschi, S., Bouquet, P., Giacomuzzi, D., Guerra, F., Po, L., Vincini, M.: An incremental method for the lexical annotation of domain ontologies. Int. J. Semant. Web Inf. Syst. **3**(3), 57–80 (2007) 866  
867  
868
5. Bergamaschi, S., Domnori, E., Guerra, F., Lado, R.T., Velegrakis, Y.: Keyword search over relational databases: a metadata approach. In: Sellis T.K., Miller R.J., Kementsietsidis A., Velegrakis Y. (eds.) SIGMOD Conference, pp. 565–576. ACM, New York (2011) 869  
870  
871
6. Bergamaschi, S., Domnori, E., Guerra, F., Orsini, M., Lado, R.T., Velegrakis, Y.: Keymantic: semantic keyword-based searching in data integration systems. PVLDB **3**(2), 1637–1640 (2010) 872  
873  
874
7. Bergamaschi, S., Guerra, F., Rota, S., Velegrakis, Y.: A hidden markov model approach to keyword-based search over relational databases. In: to appear in ER. Springer (LNCS) (2011) 875  
876
8. Bergamaschi, S., Sartori, C., Guerra, F., Orsini, M.: Extracting relevant attribute values for improved search. IEEE Inter. Comput. **11**(5), 26–35 (2007) 877  
878
9. Bergman, M.K.: The deep web: surfacing hidden value. J. Electron. Publ. **7**(1) (2001). URL <http://dx.doi.org/10.3998/3336451.0007.104> 879  
880
10. Bleiholder, J., Naumann, F.: Data fusion. ACM Comput. Surv. **41**(1) (2008) 881
11. Bourgeois, F., Lassalle, J.C.: An extension of the Munkres algorithm for the assignment problem to rectangular matrices. Commun. ACM **14**(12), 802–804 (1971) 882  
883
12. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Comput. Networks **30**(1–7), 107–117 (1998) 884  
885
13. Burkard, R., Dell'Amico, M., Martello, S.: Assignment problems. SIAM society for industrial and applied mathematics, Philadelphia (2009) 886  
887
14. Chakrabarti, S., Sarawagi, S., Sudarshan, S.: Enhancing search with structure. IEEE Data Eng. Bull. **33**(1), 3–24 (2010) 888  
889
15. Cilibrasi, R., Vitányi, P.M.B.: The google similarity distance. IEEE Trans. Knowl. Data Eng. **19**(3), 370–383 (2007) 890  
891
16. Cohen, W.W., Ravikumar, P.D., Fienberg, S.E.: A comparison of string distance metrics for name-matching tasks. IIWeb, pp. 73–78 (2003) 892  
893
17. Florescu, D., Kossmann, D., Manolescu, I.: Integrating keyword search into xml query processing. BDA (2000) 894  
895
18. Haofen, W., Kang Zhang, Q.L., Tran, D.T., Yu, Y.: Q2semantic: a lightweight keyword interface to semantic search. Proceedings of the 5th European Semantic Web Conference, LNCS, pp. 584–598. Tenerife, Spain (2008) 896  
897  
898
19. Hristidis, V., Papakonstantinou, Y.: Discover: keyword search in relational databases. VLDB, pp. 670–681 (2002) 899  
900
20. Konstanze, U., Roder, M., Hamzaoui, R.: Fast list viterbi decoding and application for source-channel coding of images. Konstanzer schriften in mathematik und informatik, <http://www.inf.uni-konstanz.de/Preprints/preprints-all.html>, pp. 801–804 (2002) 901  
902  
903
21. Kotidis, Y., Marian, A., Srivastava, D.: Circumventing data quality problems using multiple join paths. CleanDB (2006) 904  
905
22. Kumar, R., Tomkins, A.: A characterization of online search behavior. IEEE Data Eng. Bull. **32**(2), 3–11 (2009) 906  
907
23. Lam, T.Y., Meyer, I.M.: Efficient algorithms for training the parameters of hidden markov models using stochastic expectation maximization (em) training and viterbi training. Algorithms Mol. Biol. **5**(38) (2010). DOI 10.1186/1748-7188-5-38 908  
909  
910

- AQ7
24. Lember, J., Koloydenko, A.: Adjusted viterbi training. *Probab. Eng. Inf. Sci.* **21**, 451–475 (2007). DOI 10.1017/S0269964807000083. URL <http://portal.acm.org/citation.cfm?id=1291117.1291125> 911  
912  
913
  25. Li, L., Shang, Y., Shi, H., Zhang, W.: Performance evaluation of hits-based algorithms. Communications, internet, and information technology, pp. 171–176 (2002) 914  
915
  26. Li, Y., Yu, C., Jagadish, H.V.: Schema-free XQuery. VLDB, pp. 72–83 (2004) 916
  27. Liu, F., Yu, C.T., Meng, W., Chowdhury, A.: Effective keyword search in relational databases. SIGMOD, pp. 563–574. ACM, New York (2006) 917  
918
  28. Madhavan, J., Ko, D., Kot, L., Ganapathy, V., Rasmussen, A., Halevy, A.: Google's deep web crawl. *Proc. Very Large Databases (VLDB) Endow.* **1**(2), 1241–1252 (2008). DOI <http://doi.acm.org/10.1145/1454159.1454163>. URL <http://portal.acm.org/citation.cfm?id=1454163> 920  
921
  29. Maier, D., Ullman, J.D., Vardi, M.Y.: On the foundations of the universal relation model. *ACM Trans. Database Syst.* **9**(2), 283–308 (1984) 922  
923
  30. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: a versatile graph matching algorithm and its application to schema matching. ICDE, pp. 117–128. IEEE Computer Society, Silver Spring, MD (2002) 924  
925  
926
  31. Mena, E.: OBSERVER: an approach for query processing in global information systems based on interoperation across pre-existing ontologies, University of Zaragoza, 1998 927  
928
  32. Nandi, A., Jagadish, H.V.: Assisted querying using instant-response interfaces. SIGMOD, pp. 1156–1158. ACM, New York (2007) 929  
930
  33. Popa, L., Velegrakis, Y., Miller, R.J., Hernandez, M.A., Fagin, R.: Translating web data. VLDB, pp. 598–609 (2002) 931  
932
  34. Pound, J., Paparizos, S., Tsaparas, P.: Facet discovery for structured web search: a query-log mining approach. SIGMOD conference, pp. 169–180. ACM, New York (2011) 933  
934
  35. Pu, K.Q.: Keyword query cleaning using hidden markov models. In: Özsü, M.T., Chen, Y., 0002, L.C. (eds.) KEYS, pp. 27–32. ACM, New York (2009) 935  
936
  36. Qin, L., Yu, J.X., Chang, L.: Keyword search in databases: the power of rdbms. SIGMOD, pp. 681–694. ACM, New York (2009) 937  
938
  37. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. **10**(4), 334–350 (2001) 939  
940
  38. Seshadri, N., Sundberg, C.E.: List Viterbi decoding algorithms with applications. *IEEE Trans. Commun.* **42**(234), 313–323 (1994). DOI 10.1109/TCOMM.1994.577040 941  
942
  39. Simitsis, A., Koutrika, G., Ioannidis, Y.E.: Précis: from unstructured keywords as queries to structured databases as answers. VLDB J. **17**(1), 117–149 (2008) 943  
944
  40. Singhal, A., Buckley, C., Mitra, M.: Pivoted document length normalization. SIGIR, pp. 21–29 (1996) 945  
946
  41. Tata, S., Lohman, G.M.: Sqak: doing more with keywords. In: Wang J.T.L. (ed.) Proceedings of the ACM SIGMOD International Conference on Management of data, SIGMOD 2008, Vancouver, BC, Canada, pp. 889–902. ACM, New York (2008) 947  
948  
949
  42. Tata, S., Lohman, G.M.: SQAK: doing more with keywords. SIGMOD, pp. 889–902. ACM, New York (2008) 950  
951
  43. Theobald, M., Bast, H., Majumdar, D., Schenkel, R., Weikum, G.: TopX: efficient and versatile top-k query processing for semistructured data. VLDB J. **17**(1), 81–115 (2008) 952  
953
  44. Tran, T., Mathäß, T., Haase, P.: Usability of keyword-driven schema-agnostic search. 7th extended semantic web conference (ESWC'10), Greece. Springer, Berlin, Heidelberg, New York (2010) 954  
955  
956
  45. Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. ICDE, pp. 405–416. IEEE Computer Society, Silver Spring, MD (2009). DOI <http://dx.doi.org/10.1109/ICDE.2009.119> 957  
958  
959
  46. Trillo, R., Gracia, J., Espinoza, M., Mena, E.: Discovering the semantics of user keywords. J. UCS **13**(12) (2007) 960  
961
  47. Wright, A.: Searching the deep web. Commun. ACM **51**, 14–15 (2008). DOI 10.1145/1400181.1400187 962  
963

48. Yu, J.X., Qin, L., Chang, L.: Keyword Search in Databases. Morgan and Claypool, San Francisco (2010) 964  
965
49. Yu, J.X., Qin, L., Chang, L.: Keyword search in databases. Synthesis Lectures on Data Management. Morgan and Claypool, San Francisco (2010) 966  
967
50. Zenz, G., Zhou, X., Minack, E., Siberski, W., Nejdl, W.: From keywords to semantic queries-incremental query construction on the semantic web. *J. Web Semant.* **7**(3), 166–176 (2009). DOI <http://dx.doi.org/10.1016/j.websem.2009.07.005> 968  
969  
970

UNCORRECTED PROOF

AUTHOR QUERIES

- AQ1. Kindly check the corresponding author.
- AQ2. Please check if the sentence “...represents the data values in the column...” reads fine.
- AQ3. Please update “ $VW_i$  w bed.”
- AQ4. Please check if the sentence “The fact that their...” reads fine.
- AQ5. Please check the sentence “For instance, “Pearson Bill,...”.
- AQ6. Please check the sentence “The probabilities of values...”.
- AQ7. Ref. [35] is not cited in the text. Kindly cite the same or delete it from the reference list.

UNCORRECTED PROOF

## Chapter 7

# Keyword-Based Search over Semantic Data

Klara Weiand, Andreas Hartl, Steffen Hausmann,  
Tim Furche, and François Bry

### 7.1 Introduction

For a long while, the creation of Web content required at least basic knowledge of Web technologies, meaning that for many Web users, the Web was de facto a read-only medium. This changed with the arrival of the “social Web,” when Web applications started to allow users to publish Web content without technological expertise. Here, content creation is often an inclusive, iterative, and interactive process. Examples of social Web applications include blogs, social networking sites, as well as many specialized applications, for example, for saving and sharing bookmarks and publishing photos.

Social *semantic* Web applications are social Web applications in which knowledge is expressed not only in the form of text and multimedia but also through informal to formal annotations that describe, reflect, and enhance the content. These annotations often take the shape of RDF graphs backed by ontologies, but less formal annotations such as free-form tags or tags from a controlled vocabulary may also be available.

Wikis [29] are one example of social Web applications for collecting and sharing knowledge. They allow users to easily create and edit documents, so-called wiki pages, using a Web browser. The pages in a wiki are often heavily interlinked, which makes it easy to find related information and browse the content.

---

AQ1

K. Weiand (✉) · A. Hartl · S. Hausmann · F. Bry

Ludwig-Maximilians-Universität München, Oettingenstr. 67, 80538 München, Germany  
e-mail: [klara.weiand@ifi.lmu.de](mailto:klara.weiand@ifi.lmu.de); [andreas-hartl@gmx.de](mailto:andreas-hartl@gmx.de); [steffen.hausmann@ifi.lmu.de](mailto:steffen.hausmann@ifi.lmu.de);  
[bry@lmu.de](mailto:bry@lmu.de)

T. Furche

Department of Computer Science and Institute for the Future of Computing,  
Oxford University, Wolfson Building, Parks Road, Oxford OX1 3QD, UK  
e-mail: [tim@furche.net](mailto:tim@furche.net)

Semantic wikis [43] are wikis that also offer—more or less sophisticated—<sup>24</sup> formal languages for expressing knowledge as machine-processable annotations<sup>25</sup> to wiki pages. In traditional wikis, knowledge is given in the form of text in<sup>26</sup> natural language, and is not directly amenable to automated semantic processing.<sup>27</sup> Information can therefore only be located through full-text keyword search or via<sup>28</sup> simple, mostly user-generated, structures like tables of content and links between<sup>29</sup> pages. More sophisticated functionalities such as querying, reasoning, and semantic<sup>30</sup> browsing are not available. The goal behind semantic wikis is to provide at least<sup>31</sup> some of these enhancements by relying on semantic technologies, that is, knowledge<sup>32</sup> representation formalisms and methods for automated reasoning.<sup>33</sup>

To be able to leverage the knowledge contained in rich data repositories such<sup>34</sup> as semantic wikis and other social semantic applications, a query language for<sup>35</sup> social semantic Web applications should be expressive enough to allow for precise<sup>36</sup> selections using complex criteria and to enable the aggregation and combination<sup>37</sup> of data, and thus the derivation of new data through a simple form of reasoning.<sup>38</sup> Automation in the form of embedded queries (queries that are contained in a piece<sup>39</sup> of content and are evaluated when this content is retrieved) and continuous queries<sup>40</sup> (queries that are evaluated repeatedly at set intervals or when the data change)<sup>41</sup> further requires query evaluation to operate without the need for human intervention.<sup>42</sup>

Making it easy for nonexperts to publish data on the Web is an important<sup>43</sup> achievement of the social Web and a primary goal of the social semantic Web.<sup>44</sup> The goal of making the data thus produced easily accessible in turn has received<sup>45</sup> relatively little attention. This is problematic because users are likely to be less<sup>46</sup> motivated to participate in the creation of content if they cannot leverage the data<sup>47</sup> that they and others have contributed and the exploitation of the data is reserved to<sup>48</sup> expert users. The success of a social semantic Web application crucially depends on<sup>49</sup> the active participation and the contributions of its users, most of which cannot and<sup>50</sup> should not be expected to have much experience with query languages.<sup>51</sup>

Data retrieval in semantic wikis and other social (semantic) Web applications<sup>52</sup> is currently realized through keyword search or Web query languages. Keyword<sup>53</sup> search is the prevalent paradigm for search on the Web. Its strength, and presumably<sup>54</sup> the main reason for its success, is that it is very accessible: there is no syntax<sup>55</sup> that has to be learned before queries can be issued, and relevant information can<sup>56</sup> be found without any knowledge of the structure of the underlying data. On the<sup>57</sup> downside, keyword search is inherently imprecise and inexpressive. It does not<sup>58</sup> allow for the specification of structure-based selection criteria, and often not even<sup>59</sup> for logical operations. As a consequence, queries remain vague. Even when users<sup>60</sup> know precisely which data they are interested in, they may not be able to express<sup>61</sup> the corresponding selection criteria merely through keywords.<sup>62</sup>

Web query languages are in many respects the exact opposite of keyword search:<sup>63</sup> similar to queries on relational databases, Web queries are highly specific and<sup>64</sup> select individual data items which can then be processed further to reformat the<sup>65</sup> data or deduce and display new knowledge. Once defined, these tasks can be<sup>66</sup> performed automatically and without human intervention. Web query languages<sup>67</sup> are comparable to programming languages both in their expressive power and their<sup>68</sup>

complexity of use. A high cognitive investment is required before a user can employ 69  
a Web query language to retrieve data from a given dataset: in addition to the 70  
schema, the user has to know and understand the data types involved as well as 71  
the query language itself. Especially for casual or beginning Web users, acquiring 72  
this knowledge can be a hard and laborious process, and many may lack the time, 73  
dedication, motivation, or confidence to tackle it. 74

In summary, keyword search is generally more appropriate for search over 75  
weakly structured or unstructured text, while Web query languages are well suited 76  
for querying structured data. In a social semantic Web application, one typically 77  
finds both types of data. None of the methods currently available provides both a 78  
sufficient level of expressiveness and ease of use. 79

This article describes the design and implementation of KWQL, a query language 80  
for the semantic wiki KiWi. KWQL allows for rich combined queries over textual 81  
content, metadata, document structure, and informal to formal semantic annotations. 82  
The language combines keyword search and Web querying to enable a form of 83  
querying that adapts to the user's information need and knowledge and accommo- 84  
dates simple search and complex selections alike. A novel aspect of KWQL is that it 85  
combines both paradigms, keyword search and Web queries, in a bottom-up fashion. 86  
It treats neither of the two as an extension to the other, but instead integrates both in 87  
one framework. Depending on the user's knowledge and query intent, the language 88  
can behave more like keyword search or more like Web querying. 89

While querying the semantic wiki KiWi [44] is the main focus of this chapter, 90  
the underlying ideas apply more generally to querying and search on the social 91  
and social semantic Web. As such, the concepts of KWQL could be transferred 92  
to derive similar languages targeting other social semantic applications, and we 93  
consider KWQL to be exemplary of a novel family of query languages. 94

The remainder of this chapter is structured as follows: Sect. 7.2 introduces wikis 95  
and semantic wikis and gives an overview over the state of the art in querying in 96  
semantic wikis and keyword querying of semistructured data. The next section, 97  
Sect. 7.3, describes the KiWi wiki and its conceptual model. Section 7.4 then 98  
introduces KWQL and its syntax, and Sect. 7.5 gives a relational semantics for the 99  
language. visKWQL, KWQL's visual rendering, is described in Sect. 7.6. A first user 100  
evaluation of KWQL and visKWQL is described in Sect. 7.7. The following section, 101  
Sect. 7.8, describes KWilt, KWQL's patchwork-based implementation, and gives the 102  
results of a performance evaluation. 103

## 7.2 State of the Art

104

This section introduces wikis and semantic wikis and gives an overview of the 105  
search and querying functionalities provided by current semantic wiki engines. 106  
We further summarize recent research on keyword search over semistructured data. 107

### 7.2.1 Wikis: Collaborative Content Creation

108

In many respects, wikis are a prototypical social Web application, and their success 109  
is tightly connected to the proliferation of the social Web: wikis are conceptually 110  
simple, easy to use, and support users in the content creation process. 111

Apart from the original WikiWikiWeb<sup>1</sup> wiki engine, there exist a large number 112  
of wiki engines differing in their features, implementation, and application area, for 113  
example, MediaWiki,<sup>2</sup> Atlassian Confluence,<sup>3</sup> and PhpWiki.<sup>4</sup> 114

The basic elements of the conceptual model of a wiki are wiki pages and links 115  
between them. Creating or editing a wiki page is no harder than using a word- 116  
processing application, and content can be formatted using WYSIWYG editors 117  
or wiki markup. Wikis are particularly well suited for the collaborative, gradual 118  
creation of content, and they live from user participation: a wiki page may start out 119  
as a short outline and grow and evolve as more people participate or more details 120  
become known. A typical wiki page is edited and enhanced repeatedly, meaning 121  
that a final, definite version does not necessarily exist, but that each wiki page is a 122  
perpetual work in progress. 123

At the same time, wikis as knowledge management applications could profit 124  
from improved methods for structuring knowledge, making it more accessible 125  
AQ2 and amenable to automatic processing. As mentioned above, wiki pages, on the 126  
one hand, are often heavily interlinked, meaning that related concepts are often 127  
connected. In terms of structuring knowledge, this is a valuable contribution. 128  
Individual wiki pages, on the other hand, are often weakly structured and only 129  
express knowledge as free text or multimedia. 130

The term “semantic wiki” is used to refer to two different types of systems 131  
[28, 43]: semantic wikis of the first type (“wikitology” [24] or “wikis for semantics”) 132  
use wiki technology as a means for the collaborative authoring of ontologies. The 133  
main focus here is on creating semantic Web data, and human-readable wiki content 134  
is only needed to support the editing process. When used in the second sense, 135  
“semantic wiki” refers to a wiki that uses (social) semantic Web technologies 136  
to enhance the functionality of the wiki and support the process of collaborative 137  
content creation (“semantics for wikis”). Here, the focus is not (only) on metadata 138  
but (also) on text and multimedia content. Some semantic wiki engines fall clearly 139  
into one of these two categories, while others can be used for both purposes [43]. In 140  
the following, we use “semantic wiki” in the second meaning. 141

Semantic wikis extend conventional wikis by providing functionalities for 142  
expressing knowledge in a structured form. This is realized mainly by adding 143  
support for annotations to data items, most frequently wiki pages and tags, but also 144

---

<sup>1</sup><http://c2.com/cgi/wiki/>

<sup>2</sup><http://www.mediawiki.org/>

<sup>3</sup><http://www.atlassian.com/software/confluence/>

<sup>4</sup><http://phpwiki.sourceforge.net/>

smaller portions of text [23]. The annotations may be freely chosen tags [12], but sometimes, more formal mechanisms such as RDF backed by (imported) RDFS or OWL ontologies are offered as well. In particular, several semantic wikis support limited RDF annotations where the subject is always the URI of the annotated resource, and predicate and object are provided by the user [3, 4, 47].

The annotations, whether they have been assigned manually or extracted (semi) automatically, may be used for realizing functionalities like consistency checking, improved navigation, search, querying, personalization, context-dependent presentation, and reasoning. Annotations are often represented in RDF. They can thus be exported and integrated with data from other sources and are compatible with standard RDF technologies such as SPARQL.

The annotation of wiki content is optional, and semantic wikis do not require users to add annotations. While in particular only some of the users may actually annotate content, this can still enable all users of the semantic wiki to benefit from the functionalities that semantic wikis offer over conventional wikis, for example, an automatically generated table of contents [43]. Furthermore, the semantic wiki data may be formalized in a collaborative fashion over time, with different users providing the textual content and informal and formal annotations. This holds especially when different modes of annotations are available, for example, free-form tags and RDF. Semantic wikis thus maintain, at least to some extent, the ease of use of conventional wikis.

## 7.2.2 Searching and Querying in Semantic Wikis

166

Better search and querying is one of the main ways in which semantic wikis intend to improve upon conventional wikis. The need for simple yet powerful data retrieval [2, 40] and for combined queries over content and annotations [3] has been pointed out in particular. So far, however, all semantic wikis that we are aware of treat the querying of content and annotations separately [40, 43], while other sources of data such as content structure and system metadata cannot be queried at all.

In many cases, semantic wikis provide simple full-text search for the querying of textual content or RDF literals [23, 39, 47]. In addition, a standard RDF query language such as SPARQL or RDQL can often be used for querying the annotations [1, 2, 12, 42]. A number of semantic wikis also come with their own language for querying annotations that can be used in addition to or instead of a conventional RDF query language:

- KAON, the query language of COW [13], can make use of simple reasoning to find query answers.
- Rhizome [46] and its query language RxPath aim at making RDF querying easy for users who are already familiar with XML. To this end, RDF triples are mapped to a virtual, possibly infinitely recursive tree which can then be queried with XPath expressions.

- WikSAR [4] uses queries consisting of a series of predicate–object pairs. The answer to such a query then consists of all wiki pages whose annotations match all predicate–object conditions. Predicate and object can be connected by operators for equality, ranges, and regular expressions. 185  
186  
187  
188
- Two different query languages have been suggested for Semantic MediaWiki. 189  
The first, referred to as “SMW-QL” by Bao et al. [6], has a syntax similar to that 190  
used to express annotations in SMW. SMW-QL supports subqueries, (implicit) 191  
conjunction, disjunction, negation, and comparison operators, but not variables. 192  
By default, queries return wiki pages, but so-called print requests can be used to 193  
display specific property values in the query answers. Krötzsch and Vrandečić 194  
[25] provide a semantics for SMW-QL through a translation to DL queries; 195  
Bao et al. [6] define a semantics that is based on the translation of SMW-QL 196  
queries into logic programs. The second query language [17] employs keyword 197  
search over RDF data (see Section 7.2.3). Users express their query intent using 198  
a number of keywords which are matched in the data using a fuzzy scheme that 199  
considers semantic and syntactic similarity and translated into SPARQL queries 200  
which are displayed to the user in a visual, table-based form. The user can then 201  
select the query that corresponds to his or her query intent and the matching 202  
entity tuples are returned. 203

AQ3

AceWiki [26] differs from all approaches discussed above in that it employs a controlled natural language, Attempto Controlled English [15] (or ACE), to represent information in the wiki. The language is a subset of English but can be translated into a variant of first-order logic, meaning that it can be understood by humans and machines alike. Consequently, there is no distinction between content and annotations in AceWiki. The authors suggest that using ACE, queries can simply be represented as questions. 204  
205  
206  
207  
208  
209  
210

Usability and expressiveness of the above query languages vary widely; however, none of the existing languages fulfills all criteria outlined in Section 7.1, namely, that it can be used without prior training, is expressive enough to allow complex selections and can be used to query not only annotations, but also content, content structure, and metadata. 211  
212  
213  
214  
215

### 7.2.3 Keyword Querying over Semistructured Data

216

When we talk about Web queries, we subsume two distinct areas of research and technology: Web search as provided, for example, by Google or Yahoo!, and database-style queries on Web data (mostly in the form of XML or RDF) as provided through languages such as XQuery or SPARQL. 217  
218  
219  
220

Where Web search allows us to operate on (nearly) all the Web, database-style Web queries operate only on a small fraction. Where Web search is limited to filtering relevant documents for human consumption, Web queries allow for the precise selection of data items in Web documents as well as their formatting, 221  
222  
223  
224

reorganization, aggregation, and the generation of new data. Where Web search 225  
can operate on all kinds of Web documents, Web queries are usually restricted to a 226  
more homogeneous collection of documents (e.g., XHTML documents or DocBook 227  
documents). Where Web search requires a human in the loop to ultimately judge the 228  
relevance of a search result, Web queries allow automated processing, aggregation, 229  
and deduction of data. Where Web search can be used by untrained users, Web 230  
queries usually require significant training to be employed effectively. 231

In the context of social semantic software, both aspects of Web queries play 232  
an essential role: we want to be able to precisely specify selection criteria for 233  
data items and automatically derive new information, operations that squarely fall 234  
into the domain of database-style Web queries. On the other hand, the essential 235  
premise of the social semantic Web is accessibility to untrained users. In this sense, 236  
a mechanism closer to Web search is needed. Web search and Web queries have 237  
mostly been treated separately in the past, but recently, this has started to change in 238  
more than one way. 239

AQ4 The most significant efforts toward combining some of the virtues of Web 240  
search, viz. being accessible to untrained users and being able to cope with vastly 241  
heterogeneous data, are keyword-based Web query languages for XML and RDF 242  
documents. These languages operate in the same setting as XQuery or SPARQL, 243  
but with an interface suitable for untrained or barely trained users instead of a 244  
complex query language. The interface is often (in label-keyword query languages) 245  
enhanced to allow not only bag-of-word queries but also some annotations to each 246  
word, most notably a context (e.g., that a term must occur as the author or title 247  
of an article). Results are excerpts of the queried documents, though the precise 248  
extent is often determined automatically rather than by the user. Thus, keyword- 249  
based query languages trade some of the precision of languages like XQuery for 250  
a more accessible interface. The yardstick for these languages becomes an easily 251  
accessible interface (or query language) that does not sacrifice the essential premise 252  
of database-style Web queries, namely, that selection and construction are precise 253  
enough to allow for automated processing of data. 254

We can distinguish three types of keyword-based query languages for structured 255  
data according to the extent to which structure can be used as a selection criterion: 256

- In keyword-only query languages, queries consist of a number of terms which 257  
are matched to the textual content of nodes in an XML or RDF document, and 258  
in some cases to node or (in the case of RDF) edge labels. Queries make no 259  
reference to the structure of the data. This category includes most keyword query 260  
languages, like XKeyword [5, 20], XRank [16], Spark [54], and XKSearch [53]. 261
- In label-keyword query languages such as XSearch [10] and XBridge [33], a 262  
query term is a label-keyword pair of the form l:k. The term matches data where 263  
a node with the label l contains, either directly or through a descendant node, text 264  
matching the associated keyword k. It is thus possible to indicate the context in 265  
which the keyword should occur. 266
- Keyword-enhanced query languages [14, 35, 45] extend traditional Web query 267  
languages with simple keyword querying. They allow for the specification of 268

structure to the extent to which it is known, but also include constructs for the 269  
use of keyword querying where it is not. Keyword-enhanced query languages 270  
constitute an extension of traditional query languages and therefore provide their 271  
full expressive power. 272

Given that (some) Web query languages also offer ways to specify queries when 273  
the user lacks knowledge about the schema, for example, through regular path 274  
expressions in XPath, one might wonder what distinguishes traditional query 275  
languages and keyword-enhanced query languages. As pointed out by Florescu et al. 276  
[14] and Schmidt et al. [45], regular path expressions are useful when the schema 277  
is not completely known to the user, but not when the user has no knowledge of the 278  
schema at all. The reason for this is that query evaluation in Web query languages 279  
is not optimized for evaluating vague queries. Furthermore, while the schema of 280  
the data may not have to be known, knowledge of the query language itself is still 281  
necessary, making Web query languages unsuitable for casual users. 282

A second, orthogonal characteristic of keyword query languages is the way they 283  
are implemented: 284

- Most keyword query languages are implemented as stand-alone systems that 285  
handle all steps of the query evaluation. 286
- Another group of keyword query languages translate the keyword queries into 287  
another query language and thus outsource the query evaluation. This category 288  
includes many RDF keyword query languages [48, 50, 54] but, to the best of our 289  
knowledge, only one XML language, XBridge [33], which translates keyword 290  
queries into XQuery. The approach of Ladwig and Tran [27] takes an exceptional 291  
position in that it tightly integrates query translation and query evaluation, and 292  
generates queries and candidate answers at the same time. 293
- Keyword-enhanced query languages finally build on existing systems by com- 294  
bining conventional query languages like XPath or XML-QL with keyword- 295  
querying techniques. 296

The majority of keyword query languages for semistructured data in the literature 297  
are concerned with keyword-only querying of XML data. Fewer proposals exist for 298  
querying RDF data, and a majority of them translate keyword queries into traditional 299  
query languages. Most XML keyword query languages, on the other hand, evaluate 300  
queries without mapping them to another query language. 301

At the same time, keyword query languages for XML usually limit themselves 302  
to the processing of tree-shaped data, that is, roughly to XML without hyperlinks. 303  
Those languages that do work on graph-shaped XML, like XRank, ignore hyperlinks 304  
during the matching and grouping process and only use them for ranking. A notable 305  
exception is SAILER [31], which models XML and HTML documents as graphs. 306  
As Schmidt et al. [45] point out, one reason for the relative lack of keyword querying 307  
for graph-shaped XML is the expected increase in complexity and thus processing 308  
time, which would be very problematic in an application area dealing with large 309  
amounts of data. 310

Similarly, the lack of RDF keyword query languages that evaluate queries directly can be attributed to the fact that RDF is graph shaped and cannot be converted into tree-shaped data as easily as XML. In addition, querying RDF poses additional challenges because of labeled edges and blank nodes. A possible way to overcome these challenges is to summarize the RDF graph into a different structure [41, 50], but this comes at the cost of partially ignoring the structure of the data and thus reducing the granularity of the query result.

For XML querying, on the other hand, the grouping of matches is of great importance, and it is a central aspect of many approaches. The reason why determining these semantic entities in structured data is so important to keyword querying is that, in contrast to traditional query languages, queries are never fully specified, and in fact often cannot be fully specified by the user. The inferred semantics are what is used to determine what constitutes a relevant result.

Various heuristics for grouping have been proposed, a large majority of which are refinements of the established concept of the lowest common ancestor (LCA) [18], the most specific element that is an ancestor to at least one match instance of each keyword. These include, for example, SLCA [53], MLCA [35], CVLCA [30], and interconnection semantics [10]. All of these approaches add constraints to LCA in order to remedy the problem of false positives in LCA and improve the grouping of matched nodes according to their semantic entities. The approaches differ in the filter that they apply to remove undesirable results from the set of LCA nodes; each of them produces a set of results that is a subset of the results obtained by applying LCA.

AQ5

On the one hand, the different heuristics for grouping aim at being universal or at least versatile; on the other hand, they are data driven and make assumptions about the relations between structure and semantics that may not be universal. Consequently, all LCA-based grouping strategies are not universally applicable and, under certain circumstances, may lead to both false positives and false negatives [49]. This raises the question to what extent it is possible to reliably deduce semantics from structural characteristics of data alone.

While most of the approaches determine the LCA or a variant thereof automatically based on keyword match instances, an alternative approach that was used in XKeyword [5, 20] but also mentioned in connection with XRank [16] and employed in keyword querying databases [8, 11] is to manually group the data into concepts and thus predefine the possible query answer components. This method uses an extra level of processing where parts of query answers are defined a priori and therefore independent of a specific query. An obvious disadvantage is that it requires users or administrators to invest time and effort to define the groupings.

A small number of very recent approaches not only group keyword matches not just based on structure, but also take the distribution of keyword matches and node types in the data into account [7, 34]. Whether these methods will solve the problems associated with LCA-based grouping remains to be seen.

An important characteristic of traditional query languages, namely, the targeted and flexible retrieval of elements, can be found only in two of the presented stand-alone keyword query languages, in that of Cohen et al. [10] and in XSeek [36, 37].

Both of these languages return the content of a node whose label is matched. 356  
However, neither of them allows for the binding of specific values to variables. 357  
Query results thus cannot be used further in construction terms. Furthermore, it is 358  
not possible in XSeek to specify explicitly that the content of a node with a specific 359  
label should be retrieved. Rather, the necessary information is inferred from the 360  
AQ6 keyword query and is therefore relatively hard to control by the user, even if he or 361  
she knows exactly which nodes he or she would like to have returned. 362

Keyword-enhanced query languages, on the other hand, allow for a more targeted 363  
selection and enable construction to varying degrees. Schmidt et al. [45] only 364  
retrieve the label of the LCA node, the approach of Florescu et al. [14] makes the 365  
granularity of the return value dependent on the specificity of the query, and schema- 366  
free XQuery allows for the binding of variables to specific nodes in an entity subtree. 367

Flexibility with respect to the data type, that is, the ability to query data in 368  
different formats, has received relatively little attention. XRank and Sailer can be 369  
used to query both XML and HTML documents, but do so mainly by treating 370  
HTML documents as unstructured text. The combined querying of XML and RDF 371  
is particularly desirable in the context on the semantic Web, where not all content of 372  
the (XML) data is necessarily represented in (e.g., RDF) metadata, or vice versa [9]. 373  
If both could be queried using a single query language, recall would be increased, 374  
and users would only have to familiarize themselves with one query language. 375

While to the best of our knowledge there are currently no systems for the 376  
combined keyword querying of XML and RDF data, a number of approaches to 377  
keyword querying are explicitly concerned with queries over HTML and XML data 378  
and relational databases [22, 32], thereby realizing data type flexibility to a certain 379  
extent. 380

## 7.3 The KiWi Wiki

381

KiWi<sup>5</sup> is a semantic wiki with extended functionality in the areas of information 382  
extraction, personalization, reasoning, and querying. KiWi relies on a simple, 383  
modular conceptual model consisting of the following building blocks: 384

*Content items* are the primary units of information in KiWi; they correspond 385  
roughly to wiki pages in other wikis, but they can be nested: a content item 386  
can include other content items. The nesting of content items then forms a 387  
tree structure. Each content item has a URI through which it is accessible 388  
and uniquely identifiable. Content items can contain fragments, links, and tags. 389  
A content item consists of text or multimedia and an optional sequence of 390  
*contained* content items. Thus, content item nesting provides a conventional 391  
structuring of documents, for example, a chapter may consist of a sequence of 392

---

<sup>5</sup><http://www.kiwi-project.eu>

sections. For reasons of simplicity, content item containment precludes any form 393  
of overlapping or of cycles, and thus, a content item can be seen as a directed 394  
acyclic graph (of content items). 395

*Text fragments* 396 are user-defined continuous portions of text within contents items.  
They can consist of a word, a sentence, or any other section of text, and can be 397  
annotated. Text fragments are useful for—especially collaborative—document 398  
editing for adding annotations like “improve transition,” “style to be polished,” 399  
or “is this correct?” While content item expresses the structure of written text, 400  
text fragments convey narratives. Fragments can be nested but do not overlap and 401  
do not span over content items. Fragments of this kind are generally desirable, 402  
but are problematic with respect to query evaluation. While content items allow 403  
the authors to create and organize their documents in a modular and structured 404  
way, the idea behind fragments is to enable the annotation of pieces of content 405  
independently of the canonical structure given through content item nestings. If 406  
content items are like chapters and sections in a book, then fragments can be seen 407  
as passages that readers mark; they are individual and linear and in that transcend 408  
the structure of the document, possibly spanning, fragment across paragraphs or 409  
sections. 410

*Links* 411 are simple hypertext links and can be used for relating content items to each 412  
other or to external Web sites. Links have a single origin, which is a content item, 413  
an anchor in this origin, and a single target, which is also a content item. Links 413  
can be annotated. 414

*Tags* 415 are metadata that can be attached to content items, fragments, and links, 416  
describing their content or properties. They can be added by users, but can also 416  
be created by the system through automatic reasoning. Two kinds of annotations 417  
are available: tags and RDF triples. Tags allow to express knowledge informally, 418  
that is, without having to use a predefined vocabulary, while RDF triples are 419  
used for formal knowledge representation, possibly using an ontology or some 420  
other application-dependent predefined vocabulary. KWQL as presented here 421  
only supports the querying of tags, but the integration of RDF query facilities 422  
is discussed in Weiand [51]. 423

Structure, within as well as between resources, plays an important role for 424  
expressing knowledge in the wiki, ranging from tags to complex graphs of links 425  
or content item containment. 426

In the following, *resources* refer to the basic concepts in the data model—content 427  
items, links, fragments, and tag assignments (in the following referred to as “tag”) 428  
and *qualifiers* refer to properties of resources like metadata and content. *Qualifier* 429  
*values*, that is, the content associated with qualifiers, are of different types depending 430  
on the type of the qualifier. Qualifiers referring to data and metadata are associated 431  
with data in the form of dates, integers, URIs, or text. Structural qualifiers, on the 432  
other hand, describe nesting and linking relationships among pairs of content items 433  
or fragments. When used in a query, they take as a value a subquery describing the 434  
linked or nested resource. 435

## 7.4 KWQL: Principles and Syntax

436

KWQL, pronounced “quickel,” is a rule-based query language that combines the characteristics of keyword search with those of Web querying in order to enable versatile querying in the KiWi wiki. The language allows for rich combined queries of textual content, metadata, document structure, and informal to formal semantic annotations. KWQL queries range from elementary and relatively unspecific to complex and fully specified (meta)data selections. 437  
438  
439  
440  
441  
442

The key principle of KWQL is that the complexity of queries increases with their expressiveness, enabling a gradual learning of the language where required. 443 Beginning users can immediately profit from using KWQL by posing basic keyword 444 queries. As users learn more about the system and the data contained in it, their 445 information needs might begin to become more complex. KWQL allows users to 446 learn the advanced features of the language bit by bit as required to realize their 447 query intents. 448  
449

KWQL does not require a specific amount of learning from the user—it is likely 450 that some users will never venture past basic keyword queries, while others may 451 only learn to use some slightly advanced constructs but not the full language. A third 452 group may invest more time, study the full syntax, and use it to write complex rules. 453 The goal for KWQL is to equally accommodate all of these users, letting them use 454 as much or as little of the language as suits their needs. 455

KWQL queries may be vague and amount to simple full-text search or take the 456 shape of selections of individual data items using precise constraints. The language 457 is designed to support both types of queries, one similar in functionality to Web 458 search and the other similar to web querying, as well as the range of queries in 459 between. 460

Full KWQL rules consist of a query body which specifies the data to be selected, 461 and an optional head indicating how this data should be processed further. In the 462 following, we will focus on query bodies, that is, pure selection queries, and exclude 463 the discussion of construction and reasoning in KWQL. 464

Query bodies can express selections of varying levels of complexity using any 465 combination of data sources in the wiki. For example, a content item selection 466 can refer not only to the textual content of the content item to be selected, but 467 also to the structuring of its contained content items, to the links from or to the 468 content item, and to its annotations. In short, KWQL is fully aware of the underlying 469 conceptual model. 470

To improve the user experience, and simplify the mental transfer of the query 471 intent into a query, query bodies take the shape of abstracted descriptions of the data 472 to be matched. This query-by-example-like syntactic style is further substantiated by 473 the fact that KWQL query terms are injective, meaning that no two query terms may 474 match the same data item. For example, when a query body describes a content item 475 with two tags, one with name “wiki” and one created by the user Mary, the query 476 will retrieve only content items where the two conditions hold for two distinct tags, 477 but not those where a single tag satisfies both criteria but no other tag meets either of 478 them. Apart from enhancing the expressive power of KWQL, injectivity also more 479

AQ7 tightly couples the user experience—what the user sees and perceives when he or she uses the wiki—to the way in which queries are expressed in KWQL. 480  
481

Each query body evaluates to a set of content items, namely, those that are 482 compatible with the given description. Compatibility here means that the content 483 item has all the properties specified in the query body and may in addition have any 484 number of other properties not in contradiction with the selection criteria. 485

KWQL's scaling with user experience and the specificity of the query intent is 486 realized through far-reaching and comprehensive support for the underspecification 487 of queries. The simplest—and at the same time most vague—description of content 488 items to be matched consists of one or several keywords that the content items must 489 contain. When the context in which the keywords may occur is not restricted further, 490 all content items that contain the given keywords in their text, title, fragments, 491 links, tags, or associated metadata—but not in linked or nested content items— 492 are compatible with the query and returned as results. Basic keyword queries in 493 KWQL therefore constitute a true full-text search over all parts of the individual 494 content items. 495

To make queries more selective and precise, the structural context in which the 496 keywords should occur can be specified fully or in part. In addition to conjunction, 497 which is implicitly assumed when no operator is given, operators for disjunction 498 and negation may be used. KWQL bodies thus amount to descriptions of the data to 499 be retrieved that, depending on the users' knowledge and information need, can be 500 more or less specific. 501

This approach lends itself particularly well to stepwise querying, the gradual 502 refinement of queries: starting with explorative queries using a small set of 503 keywords, users can go through several iterations of evaluating a query, examining 504 the results, and then further substantiating the query until the desired information is 505 found. 506

KWQL allows for the selection of data based on the structure of content items and 507 fragments through the `child` and descendant qualifiers. To keep the language 508 simple, navigational queries are avoided and no qualifiers are offered for parents 509 and ancestors. Olteanu et al. [38] have shown that adding these *backward axes* does 510 not increase the expressiveness of a query language. 511

KWQL's structural qualifiers give rise to recursive data retrieval through a wiki 512 page structure. These qualifiers take subqueries as a value, that is, arbitrary KWQL 513 queries specifying selection constraints on a linked or nested content item or 514 fragment. Structure qualifiers can thus be seen as edges to other content items or 515 fragments, and recursive querying as traversal of the resulting graph. 516

Link traversal can be expressed similarly. It should be noted that, despite the fact 517 that structural queries and link traversals can be nested, no infinite loops can occur. 518 This is due to the fact that queries are always finite and KWQL does not support 519 Kleene closure. 520

Query bodies may also contain variables. In the query evaluation process, these 521 are bound to specific values of the matching content items, for example, their 522 authors or the titles of the content items that they link to. Variables can serve three 523 purposes: 524

**Table 7.1** KWQL qualifier types

Qualifier	Resource type(s)	Value type	Arity	
<i>Data</i>				
Title	Content item	String	1	t14.2
Text	Content item; fragment	String	1	t14.3
Anchortext	Link	String	1	t14.4
Name	Tag	String	1	t14.5
<i>Metadata</i>				
URI	Content item; fragment; tag	URI	1	t14.6
Author	Content item; fragment; tag	String	+	t14.7
Created	Content item; fragment; tag	Date	1	t14.8
lastEdited	Content item	Date	1	t14.9
numberEdits	Content item	Integer	1	t14.10
<i>Structure</i>				
Child	Content item	Content item	*	t14.11
Child	Fragment	Fragment	*	t14.12
Descendant	Content item	Content item	*	t14.13
Descendant	Fragment	Fragment	*	t14.14
Target	Link	Content item	1	t14.15

- To bind values for further use in the construction part of a rule. 525
- As a wildcard or existential quantifier. 526
- To enforce that two qualifiers have identical values. In KWQL, all occurrences 527 of a variable in a query body must have the same value; using the same variable 528 several times therefore amounts to imposing equality constraints on the values of 529 the respective qualifiers. 530

KWQL supports two types of queries: regular queries, evaluated only once, and 531 embedded queries. Embedded queries are part of a content item and are evaluated 532 every time the content is loaded. They enable predefined views that always display 533 the latest information without the need for manual updating. 534

## 7.4.1 Syntax

535

Table 7.1 lists all qualifier types together with the resources in which they can 536 appear, the data type of their value, and the arity of the qualifier term. \* and + 537 here are used as in regular expressions, indicating that the qualifier can appear any 538 number of times (\*) or, arbitrarily, often but at least once (+). 539

Nonstructural qualifiers and subresources describe the *intracontent item* 540 structure. Structural qualifiers impose constraints on the *intercontent item* 541 structure and the *interfragment* structure in the case of child and descendant. 542

```

⟨kwql-query⟩ ::= ⟨resource-term⟩
⟨resource-term⟩ ::= ⟨value-term⟩ | ⟨qualifier-term⟩
| ⟨structure-term⟩
| ⟨resource-term⟩ ('OR' | 'AND')? ⟨resource-term⟩
| '(' ⟨resource-term⟩ ')'
| 'NOT' ⟨resource-term⟩
| ⟨resource⟩ '(' ⟨resource-term⟩ ')'
⟨resource⟩ ::= 'link' | 'ci' | 'fragment' | 'tag'
⟨structure-term⟩ ::= ('child' | 'descendant' | 'target') ':'
⟨resource-term⟩
⟨value-term⟩ ::= ⟨STRING⟩
⟨qualifier-term⟩ ::= ⟨qualifier⟩ ':' (⟨value-term⟩ | ⟨variable⟩)
⟨qualifier⟩ ::= 'text' | 'title' | 'name' | 'URI' | 'agree'
| 'disagree' | 'lastEdited' | 'numberEd'
| 'author' | 'created' | 'anchorText'
⟨variable⟩ ::= '$'⟨IDENTIFIER⟩

```

**Fig. 7.1** KWQL syntax**Table 7.2** KWQL example queries

Java	Content items containing “Java” directly or in any of its tags or other metadata	t15.1
ci(author:Mary)	Content items authored by Mary	t15.2
ci(Java OR (tag(XML) AND author:Mary))	Content items that either contain “Java” or have a tag containing “XML” and are authored by Mary	t15.3
ci(tag(name:\$x author:Mary) tag(name:\$x author:John))	Content items that are tagged with the same tag by both John and Mary	t15.4
ci(tag(episode) tag(name:like author:Mary) tag(name:like author:John))	“Episode” content items that both Mary and John like	t15.5
ci(tag(Java) link(target:ci(Lucene)))	Content items with a tag containing “Java” that contain a link to a content item containing “Lucene”	t15.6
ci(URI:\$a tag(character) link(target:ci( tag(location) link(target:ci(URI: \$a))))))	Character content items that link to a location content item that links back to them	t15.7

A (somewhat simplified) grammar for KWQL query bodies is given in Figure 7.1. Examples of KWQL queries together with their natural language translations are shown in Table 7.2.

## 7.5 Semantics

546

For defining a formal semantics of KWQL, we introduce an abstraction of the data model of KWQL, called KWQL graphs.

547

548

**Definition 7.1 (KWQL Graph).** Let  $\mathcal{Q} = \{text, title, \dots\}$  be the set of all  $n$  KWQL 549 qualifiers and  $\mathcal{V}$  the set of all qualifier values. Then, a *KWQL graph* is an  $(n + 6)$ - 550 tuple  $G = (\mathcal{C}, \mathcal{F}, \mathcal{L}, \mathcal{T}, S, C, Q_{\lambda_1}, \dots, Q_{\lambda_n})$ , where 551

- $\mathcal{C}$  is the set of all content items (wiki pages) 552
- $\mathcal{F}$  is the set of all fragments 553
- $\mathcal{L}$  is the set of all links 554
- $\mathcal{T}$  is the set of all tags 555
- $\mathcal{R} := \mathcal{C} \uplus \mathcal{L} \uplus \mathcal{T} \uplus \mathcal{F}$  is the set of all resources 556
- $S \subset (\mathcal{C} \times \mathcal{C}_<) \cup (\mathcal{F} \cup \mathcal{F}_<) \cup (\mathcal{L} \cup \mathcal{L}_<)$  is the association relation between resources 557 where  $\mathcal{C}_< = \mathcal{F} \cup \mathcal{L} \cup \mathcal{T}$ ,  $\mathcal{F}_< = \mathcal{L} \cup \mathcal{T}$ ,  $\mathcal{L}_< = \mathcal{T} \cup \mathcal{C} \cup \mathcal{F}$  558
- $C \subset \mathcal{C} \times (\mathcal{C} \cup \mathcal{F})$  is the containment relation between wiki pages and fragments 559 and  $C^+ = \bigcup_{n \geq 1} C^n$  is the transitive closure of  $C$  560
- For each qualifier  $\lambda \in \mathcal{Q}$ ,  $Q_\lambda \subset \mathcal{R} \times \mathcal{V}$  associates the values for  $\lambda$  to a KWQL 561 resource 562

The KWQL semantics is defined based on KWQL graphs and given in Table 7.3 563 in terms of three functions,  $\llbracket \cdot \rrbracket_{ci}$ ,  $\llbracket \cdot \rrbracket$ , and  $\llbracket \cdot \rrbracket_{dir}$ . A KWQL query is constrained by  $\llbracket \cdot \rrbracket_{ci}$  564 to return only content items (i.e., elements of  $\mathcal{C}$ ). Most expressions can occur in two 565 contexts, represented by the semantic functions  $\llbracket \cdot \rrbracket$  and  $\llbracket \cdot \rrbracket_{dir}$ : In the first context, a 566 query such as Java returns all resources that contain “Java” directly in any of their 567 qualifiers or indirectly in the qualifiers of any of their fragments, tags, and links. 568 In the second context, only resources that contain “Java” directly are returned. The 569 exception to this rule are keyword queries which are always interpreted in the first 570 manner. 571

The semantics in Table 7.3 handles variables, but omits the injectivity constraints 572 for readability reasons. To handle variables, we introduce the set  $\mathcal{I}$  of KWQL 573 variables and the set  $\mathcal{B} = 2^{\mathcal{I} \times \mathcal{V}}$  of possible variable assignments (pairs of variables 574 and value). We further extend the set operators  $\cup$  and  $\cap$  to pairs of resources and 575 variable assignments as follows: Let  $A, B \in 2^{\mathcal{R} \times \mathcal{B}}$ . 576

Then  $A \cap B = \{(r, \beta) \in \mathcal{R} \times \mathcal{B} : (r, \beta') \in A \wedge (r, \beta'') \in B \wedge \beta = \beta' \cap \beta'' \wedge \beta \neq \emptyset\}$ , 577  $A \cup B = \{(r, \beta' \cup \beta'') \in \mathcal{R} \times \mathcal{B} : (r, \beta') \in A \wedge (r, \beta'') \in B\} \cup \{(r, \beta') \in \mathcal{R} \times \mathcal{B} : 578 (r, \beta') \in A \wedge \exists \beta'' : (r, \beta'') \in B\} \cup \{(r, \beta') \in \mathcal{R} \times \mathcal{B} : (r, \beta') \in B \wedge \exists \beta'' : (r, \beta'') \in A\}$ . 579

## 7.6 visKWQL

This chapter describes visKWQL (Hartl et al. [19]), a visual rendering of KWQL 581 that allows to expressing queries using a visual formalism. 582

visKWQL fully supports KWQL in the sense that every KWQL query can 583 be expressed as an equivalent visKWQL query and vice versa. In order to avoid 584 introducing additional constructs and thus additional complexity, the rendering 585 stays close to the textual language in its visual representation: visKWQL uses a 586 form-based approach. All KWQL elements, including resources, qualifiers, and 587 operators, are represented as boxes. Resource-value or qualifier-value associations 588

**Table 7.3** Semantics for KWQL

$\llbracket \langle kwql-query \rangle \rrbracket_{ci}$	$= \pi_1(\llbracket \langle kwql-query \rangle \rrbracket(\emptyset)) \cap C$
$\llbracket \langle STR \rangle \rrbracket_{dir}(\beta) = \llbracket \langle STR \rangle \rrbracket(\beta)$	$= \{(r, \beta) \in \mathcal{R} \times \mathcal{B} : \exists \lambda, v : Q_\lambda(r, v) \wedge \text{contains}(v, \langle STR \rangle)\} \cup$ $\{(r, \beta) : \exists r' \in \mathcal{R} : S(r, r') \wedge (r', \beta) \in \llbracket \langle STR \rangle \rrbracket(\beta)\}$
$\llbracket \langle qualifier \rangle^* : \langle STRING \rangle \rrbracket_{dir}(\beta)$	$= \{(r, \beta) \in \mathcal{R} \times \mathcal{B} : Q_{\langle qualifier \rangle}(r, v) \wedge \text{contains}(v, \langle STRING \rangle)\}$
$\llbracket \langle qualifier \rangle^* : \langle STRING \rangle \rrbracket(\beta)$	$= \llbracket \langle qualifier \rangle^* : \langle STRING \rangle \rrbracket_{dir}(\beta) \cup$ $\{(r, \beta) : \exists r' \in \mathcal{R} : S(r, r') \wedge (r', \beta) \in \llbracket \langle qualifier \rangle^* : \langle STRING \rangle \rrbracket(\beta)\}$
$\llbracket \langle qualifier \rangle^* : '$(IDENT)' \rrbracket_{dir}(\beta)$	$= \{(r, \beta) \cup \{((IDENT), v)\} \in \mathcal{R} \times \mathcal{B} : Q_{\langle qualifier \rangle}(r, v) \wedge$ $(\exists v' : ((IDENT), v') \in \beta \vee ((IDENT), v) \in \beta)\}$
$\llbracket \langle qualifier \rangle^* : '$(IDENT)' \rrbracket(\beta)$	$= \llbracket \langle qualifier \rangle^* : \langle STRING \rangle \rrbracket_{dir}(\beta) \cup$ $\{(r, \beta) : \exists r' \in \mathcal{R} : S(r, r') \wedge (r', \beta) \in \llbracket \langle qualifier \rangle^* : '$(IDENT)' \rrbracket(\beta)\}$
$\llbracket \langle resource \rangle^* : \langle res-term \rangle \rrbracket_{dir}(\beta)$	$= \{(r, \beta') \in \mathcal{R} \times \mathcal{B} : \text{type}(r, \langle resource \rangle) \wedge (r, \beta') \in \llbracket \langle res-term \rangle \rrbracket_{dir}\}$
$\llbracket \langle resource \rangle^* : \langle res-term \rangle \rrbracket(\beta)$	$= \llbracket \langle resource \rangle^* : \langle res-term \rangle \rrbracket_{dir}(\beta) \cup$ $\{(r, \beta) : \exists r' \in \mathcal{R} : S(r, r') \wedge (r', \beta) \in \llbracket \langle resource \rangle^* : \langle res-term \rangle \rrbracket(\beta)\}$
$\llbracket \cdot \text{child}^* : \langle kwql-query \rangle \rrbracket(\beta)$	$= \{(r, \beta') \in (C \cup \mathcal{F}) \times \mathcal{B} : \exists r' \in \mathcal{R} : C(r, r') \wedge (r', \beta') \in \llbracket \langle kwql-query \rangle \rrbracket\}$
$\llbracket \cdot \text{descendant}^* : \langle kwql-query \rangle \rrbracket(\beta)$	$= \{(r, \beta') \in (C \cup \mathcal{F}) \times \mathcal{B} : \exists r' \in \mathcal{R} : C^+(r, r') \wedge (r', \beta') \in \llbracket \langle kwql-query \rangle \rrbracket\}$
$\llbracket \cdot \text{target}^* : \langle kwql-query \rangle \rrbracket(\beta)$	$= \{(r, \beta') \in \mathcal{L} \times \mathcal{B} : \exists r' \in \mathcal{R} : S(r, r') \wedge (r', \beta') \in \llbracket \langle kwql-query \rangle \rrbracket\}$
$\llbracket \langle res-term \rangle_1 \langle res-term \rangle_2 \rrbracket(\beta)$	$= \llbracket \langle res-term \rangle_1 \rrbracket(\beta) \sqcap \llbracket \langle res-term \rangle_2 \rrbracket(\beta)$
$\llbracket \langle res-term \rangle_1 \cdot \text{AND} \cdot \langle res-term \rangle_2 \rrbracket(\beta)$	$= \llbracket \langle res-term \rangle_1 \rrbracket(\beta) \sqcap \llbracket \langle res-term \rangle_2 \rrbracket(\beta)$
$\llbracket \langle res-term \rangle_1 \cdot \text{OR} \cdot \langle res-term \rangle_2 \rrbracket(\beta)$	$= \llbracket \langle res-term \rangle_1 \rrbracket(\beta) \sqcup \llbracket \langle res-term \rangle_2 \rrbracket(\beta)$
$\llbracket \cdot (\langle res-term \rangle^*) \cdot \rrbracket(\beta)$	$= \llbracket \langle res-term \rangle \rrbracket(\beta)$
$\llbracket \cdot \text{NOT}^* \cdot (\langle res-term \rangle^*) \cdot \rrbracket(\beta)$	$= \mathcal{R} \setminus \pi_1(\llbracket \langle res-term \rangle \rrbracket(\beta)) \times \{\beta\}$
$\llbracket \langle res-term \rangle_1 \langle res-term \rangle_2 \rrbracket_{dir}(\beta)$	$= \llbracket \langle res-term \rangle_1 \rrbracket_{dir}(\beta) \sqcap \llbracket \langle res-term \rangle_2 \rrbracket_{dir}(\beta)$
$\llbracket \langle res-term \rangle_1 \cdot \text{AND} \cdot \langle res-term \rangle_2 \rrbracket_{dir}(\beta)$	$= \llbracket \langle res-term \rangle_1 \rrbracket_{dir}(\beta) \sqcap \llbracket \langle res-term \rangle_2 \rrbracket_{dir}(\beta)$
$\llbracket \langle res-term \rangle_1 \cdot \text{OR} \cdot \langle res-term \rangle_2 \rrbracket_{dir}(\beta)$	$= \llbracket \langle res-term \rangle_1 \rrbracket_{dir}(\beta) \sqcup \llbracket \langle res-term \rangle_2 \rrbracket_{dir}(\beta)$
$\llbracket \cdot (\langle res-term \rangle^*) \cdot \rrbracket_{dir}(\beta)$	$= \llbracket \langle res-term \rangle \rrbracket_{dir}(\beta)$
$\llbracket \cdot \text{NOT}^* \cdot (\langle res-term \rangle^*) \cdot \rrbracket_{dir}(\beta)$	$= \mathcal{R} \setminus \pi_1(\llbracket \langle res-term \rangle \rrbracket_{dir}(\beta)) \times \{\beta\}$

are represented as box nestings. Boxes consist of a label, the name of the represented KWQL element, and a body, which can hold one or more child boxes. This approach has several advantages: it stays close to KWQLs textual structure, keeping visKWQL simple and making it easy to translate between the two representations; it also lends itself well to rendering in HTML. Figure 7.2 shows an example of a visKWQL query corresponding to the textual KWQL query tag(author:Mary AND name:wiki), which retrieves content item that Mary has tagged with “wiki” or that contain a fragment or link with such a tag.

An accompanying editor, the KWQL query builder (KQB, see Figure 7.3), allows for the easy and straightforward construction of queries using drag-and-drop and, in addition, supports the user during query construction by displaying tooltips, preventing syntactic errors where possible, and by pointing the user to syntactically incorrect parts of a query. All actions in the editor apart from entering text into text

**Fig. 7.2** A visKWQL query



**Fig. 7.3** The KiWi query builder

fields consist of drag-and-drop or left-click operations. There are no context menus or other interaction modes that might confuse users. 602  
603

The KQB further provides features like information hiding to only display parts 604 of larger queries and the highlighting of all occurrences of a variable when the 605 mouse pointer is positioned over a variable in a query. 606

One particularly important feature of KQB is round tripping, which allows users 607 to edit a query in both representations, visual and textual, at the same time, and 608 see any changes made to one representation reflected in the other. The side-by-side 609

display of both representations offers the additional advantage of helping users to 610  
learn KWQL by creating queries in visKWQL. Users do not have to decide in 611  
advance which formalism, textual or visual, they use to create a query, but should 612  
be able to switch between both at any time. For example, the user can start with a 613  
simple textual query, add an element to it in the visual representation, and finally 614  
edit a value in the textual representation before evaluating the query. 615

The visual KWQL query editor does not require the installation of special 616  
software or browser plug-ins, but instead is implemented using DHTML, with 617  
HTML and CSS for the presentation and Javascript for the program logic and user 618  
interaction. As a consequence, the system runs completely on the client side, within 619  
the users' Web browser, and the translation from visKWQL to KWQL can be seen 620  
as a serialization of the visual query. 621

AQ9

## 7.7 User Evaluation

622

This section describes the setup and results of a user study performed to evaluate the 623  
suitability of KWQL and visKWQL for querying tasks in the KiWi wiki. A question 624  
of particular interest is whether the results differ (1) between users with varying 625  
amounts of previous experience in the area of query languages and social semantic 626  
software and (2) between participants using textual KWQL and those using its visual 627  
rendering. 628

AQ10

The evaluation discussed here was performed as a single-session experiment 629  
where participants were given a short introduction into the KiWi wiki and, depending 630  
on the group they had been assigned to, KWQL or visKWQL was then asked 631  
to formulate queries ranging from simple and vague to precise and expressive. In a 632  
second task, participants were confronted with KWQL or visKWQL queries which 633  
they then translated into natural language descriptions of the data selected by the 634  
queries. Throughout the process, participants were encouraged to write down their 635  
thoughts and opinions on KiWi, the query language, and individual tasks. 636

However, to limit the scope of the experiment and focus on the aspects outlined 637  
above, several factors are intentionally not treated in this first evaluation. These 638  
include the gradual, self-paced learning process, user's individual query intents, 639  
long-distance effects and individual preferences for either KWQL or visKWQL. 640

### 7.7.1 Experimental Setup and Execution

641

To reflect the collaborative process of content creation and annotation, several user 642  
accounts were created in an installation of the KiWi wiki. Each account was used to 643  
compose a number of content items containing text from a wiki on the TV show The 644

Simpsons.<sup>6</sup> These content items were then annotated with tags. The final dataset, used in this study, consisted of 653 content items.

Twenty-one participants were recruited via an Internet forum aimed at LMU Munich computer science students and via announcements in several computer science lectures at the university. Sixteen of the participants were students of computer science or media computer science, one was a researcher in a nonrelated area of computer science, and four participants were students of subjects other than computer science.

Before the experiment, participants were asked to fill out a questionnaire about their previous experience in areas relevant to semantic wikis and KWQL such as the semantic Web, tagging, and XML; participants were also asked which programming and query languages they knew. Each participant was then randomly assigned to either the KWQL or the visKWQL group.

In an introductory phase, participants were allowed to familiarize themselves with the KiWi wiki and KWQL or visKWQL for 30 min. This was followed by a query creation task and a query understanding task, which, respectively, lasted 45 and 15 min. The objective of the query creation task was to use KWQL or visKWQL to answer questions, given in natural language, about the data in the wiki. In total, the task consisted of ten assignments of increasing difficulty. In the query understanding task, participants were given six KWQL or visKWQL queries of intermediate to advanced complexity, and were asked to describe the underlying query intent, that is, the common characteristics of the content items selected by each query, using natural language.

### 7.7.2 Results

Participants' self-assessed average knowledge of various areas relevant to KWQL and visKWQL was very similar for the two groups. The only concepts participants were familiar with to some extent were wikis, XML, and tags.

For the analysis, participants were divided into two groups based on their previous knowledge of query languages, social software, and semantic Web technologies. In the following, participants will often be referred to as "novice participants" or "advanced participants" based on the group they were assigned to. Depending on previous knowledge and the query language used, each participant thus belonged to one of four groups. The number of participants in each group was between four and six.

<sup>6</sup>[http://simpsons.wikia.com/wiki/Simpsons\\_Wiki](http://simpsons.wikia.com/wiki/Simpsons_Wiki)

## Task 1: Query Creation

679

Table 7.4 shows the average number of questions, out of a total of ten, answered by the participants in each group (ignoring whether the solution was correct or not). The number is higher for advanced participants (8.34) compared to novice participants (8.02), and slightly higher for KWQL users (8.20) than for visKWQL users (8.12). Furthermore, KWQL and visKWQL show reversed effects with respect to how the amount of questions answered differs with proficiency: while advanced KWQL participants on average answered 0.8 questions more than their less experienced counterparts, advanced visKWQL users answered 0.2 questions less than visKWQL novices.

Table 7.5 shows the average percentage of the given answers that were correct. Among all participants, almost two thirds of all answers given, 62.24%, were correct. The visKWQL group was responsible for both the best and the worst results, with 93.33% correct answers for advanced visKWQL users and 35.66% correct answers for novice visKWQL. This result is particularly noteworthy since both groups answered a very similar amount of questions, as shown in Table 7.4. While novice visKWQL users answered more questions on average than novice KWQL users, a smaller percentage of those answers were correct, leading to a higher absolute number of correct answers for the KWQL group. Among the advanced groups, the situation is different: visKWQL users answered fewer questions but did so at a very high rate of correctness. As a consequence, the average absolute number of correct answers is higher for advanced visKWQL users.

Out of the total of 171 queries given as answers to questions in the query creation task, only seven, four KWQL queries and three visKWQL queries, were invalid in the sense that they could not be parsed or violated a validity constraint.<sup>7</sup> Consequently, 95% of all KWQL queries and 97% of all visKWQL queries given as answers were valid. The majority of incorrect answers therefore consisted of queries that were valid but did not correspond to the assignment.

**Table 7.4** Average number of questions (out of ten) answered

	KWQL	visKWQL	Overall	t16.1
Novice	7.8	8.2	8.02	t16.2
Advanced	8.6	8.0	8.34	t16.3
Overall	8.20	8.12	8.16	t16.4

**Table 7.5** Average percentage of given answers that are correct

	KWQL	visKWQL	Overall	t17.1
Novice	53.33	35.66	43.70	t17.2
Advanced	78.17	93.33	84.91	t17.3
Overall	65.75	58.73	62.24	t17.4

<sup>7</sup>In addition, six queries were bracketed incorrectly, but since participants had to write down their answers by hand, this is likely due to clerical errors and was ignored.

**Table 7.6** Average number of questions (out of six) answered in task 2

	KWQL	visKWQL	Overall	t18.1
Novice	5.5	5.5	5.5	t18.2
Advanced	6	6	6	t18.3
Overall	5.75	5.7	5.72	t18.4

**Table 7.7** Average number of questions (out of six) answered correctly

	KWQL	visKWQL	Overall	t19.1
Novice	4.75	4.0	4.34	t19.2
Advanced	5.5	5.5	5.5	t19.3
Overall	5.13	4.6	4.89	t19.4

## Task 2: Query Understanding

707

In the query understanding task, all advanced participants provided answers to all six questions, while the novice participants answered 5.5 questions on average (see Table 7.6). Overall, participants answered 4.89 of the questions correctly. There was no difference in the number of correct answers between advanced participants who used KWQL and those who used visKWQL: both gave 5.5 correct answers on average. The situation is different for the novice users: here, those using KWQL had 4.75 correct answers on average, while participants in the visKWQL group only answered 4.0 questions correctly on average. Overall, this means that KWQL users gave more correct answers than visKWQL users by 0.53 questions, while advanced users on average answered 1.16 more questions correctly than novice users did (Table 7.7).

AQ11 718

## User Judgments

719

After completing the two tasks, participants were asked about their opinion on KWQL or visKWQL. Most participants, 13 out of 20, said that they felt they had understood how to use the respective query language. Six participants stated that they had understood the language to some extent, but had trouble with specific concepts or needed more time to understand it fully. Only one participant claimed to not have understood visKWQL at all.

With respect to the question whether KWQL or visKWQL was easy to use, a majority of participants answered that it was. However, many qualified their response and listed particular aspects they found hard to understand. Specifically, participants experienced problems with variables, URIs, injectivity, nesting of content items, and links. In several cases, participants did not understand the question or were unsure how to translate it into a query.

Finally, participants were asked what they considered to be advantages and disadvantages of KWQL and visKWQL. All participants thought that KWQL and visKWQL are powerful and allow for precise queries, while some remarked that they are harder to use than Web search and take some time to learn.

### 7.7.3 Discussion

736

All in all, the results of the experimental evaluation are very positive: KWQL 737 and visKWQL were well perceived by the participants. Given only a very short 738 introduction and a small amount of time to solve the assignments, participants 739 overall could provide correct answers to more than half the questions in the query 740 writing task and over 80% of the questions in the query understanding task. 741

The amount of learning required could explain why visKWQL novices per- 742 formed worse than the participants in the novice KWQL group: apart from having 743 to learn all the new concepts, they also had to acquaint themselves with visual 744 querying, which likely was unfamiliar to them. The novice KWQL users, on the 745 other hand, had to write textual queries, which, given that all participants can be 746 assumed to have used Web search engines before, was more familiar to them. 747

Another contributing factor to the comparatively bad performance of novice 748 visKWQL users could be that visKWQL is not ideally suited for creating vastly 749 underspecified queries. visKWQL makes it easy to understand the structure of 750 queries and to create structured queries, but offers no advantage when then 751 queries involved are very simple. Indeed, novice visKWQL participants performed 752 particularly badly compared to novice KWQL participants on questions that require 753 underspecified queries, and the difference in the percentage of correct answers was 754 smaller when the answer queries contained more structure. 755

This result indicates that it might be better to introduce beginning users whose 756 queries exclusively consist of keywords to textual KWQL, and to only add 757 visKWQL once the queries become more complex. On the other hand, given the 758 round-tripping capabilities of visKWQL, it is possible that users could achieve 759 equivalent or better results when textual and visual query editing are introduced 760 simultaneously; a follow-up study could investigate which of the three methods 761 yields the best results. 762

Advanced participants achieved good results regardless of the query language: on 763 average, they answered 71% of the questions in the query creation task and over 90% 764 of the questions in the query understanding task correctly. Their results also showed 765 that visKWQL can help to improve the performance: advanced visKWQL partic- 766 ipants gave fewer answers overall than advanced KWQL participants, but nearly 767 all of their answers were correct. These findings indicate that participants who are 768 familiar with querying and structured data and to whom the information in the 769 introductions is less novel can make effective use of visKWQL and the advantages 770 it offers over textual KWQL. This result gives further weight to the explanation that 771 the comparatively bad performance of novice visKWQL participants is due to them 772 being confronted with an overwhelming amount of new information that makes it 773 hard for them to additionally absorb the concepts of visual querying and visKWQL. 774

Across all groups, participants had more success understanding queries than writ- 775 ing them. In the query understanding task, novice KWQL users again outperformed 776 novice visKWQL users, both of which had a lower percentage of correct answers 777 than either advanced group. Both advanced groups on average answered more than 778

AQ12

90% of the questions correctly, indicating that this task was very easy for them 779 overall. The fact that participants performed better at understanding queries than at 780 writing them indicates that users could benefit from the addition of query templates 781 that users can modify according to their needs. 782

## 7.8 KWilt: Patchwork Knowledge Management

783

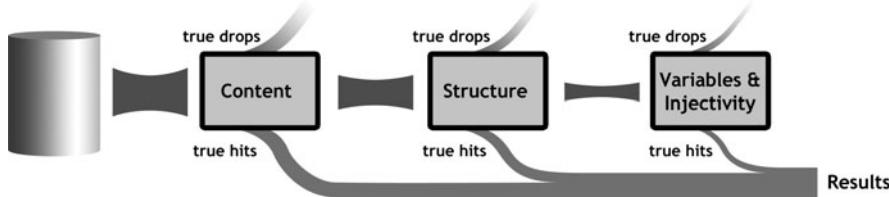
AQ13 KWilt is the implementation of KWQL in KiWi. It provides an easily extensible, 784 yet performance implementation of the KWQL features over the wide range of data 785 available in KiWi. Previous approaches have often tried to engineer a knowledge 786 information systems for such diverse information and user needs from the start. 787 By contrast, KWilt uses a “patchwork” approach, combining performance and 788 mature technologies where available. For example, KWilt uses a scalable and well- 789 established information retrieval engine to evaluate keyword queries. The patchwork 790 approach has three main advantages: 791

- AQ14
- Many queries can be evaluated at the speed of search engines, yet all the power of 792 first-order logic is available if needed. The three steps use not only increasingly 793 more expressive but also less scalable technologies. Thus, even for queries that 794 involve full first-order constraints, we can in many cases substantially reduce 795 the number of candidates in the information retrieval engine. This property is 796 particularly relevant in the context of KWQL, as (novice) users who use KWQL 797 like a search engine also expect the speed of a search engine, unaware of the 798 additional expressiveness provided by KWQL. 799
  - Each part is implemented using proven technologies and algorithms with mini- 800 mal “glue” between the employed tools. 801
  - The separation makes it easy to adapt each of the parts, for example, to reflect 802 additional data sources. If KiWi would introduce data with different structural 803 properties, for example, strictly hierarchical taxonomies in addition to RDF 804 ontologies, only the part of KWilt that evaluates structural constraints needs to 805 be modified. Similarly, if KWQL would introduce other content primitives other 806 than keywords (e.g., for image retrieval), only the first (retrieval) part of KWilt 807 would be affected. 808

### 7.8.1 Architecture and Evaluation Phases

809

Despite the unique combination of features found in KWQL, KWilt does not try 810 to “reinvent the wheel.” Instead, we used a *patchwork*, or integration, approach 811 to combine off-the-shelf state-of-the-art tools in a single framework. To this end, 812 the evaluation is split into three different evaluation phases, which are dedicated to 813 certain aspects of the query, see Figure 7.4. Each step makes use of a tool that is 814



**Fig. 7.4** The evaluation pipeline of the framework

particularly suitable for evaluating the query constraints covered by that aspect of the entire evaluation. Thus, efficient and mature algorithms form the basis of our framework.

### 7.8.1.1 Evaluation of Keyword Queries

Most KWQL queries, in particular by novice users, mainly or exclusively regard the content of the pages. Therefore, the first evaluation phase regards the keyword parts of a query in order to evaluate them in an early phase of the evaluation with as little overhead as possible. If all constraints of the query can be validated in this phase, the two subsequent phases can be skipped.

The information retrieval engine Solr provides a highly optimized inverted list index structure to carry out keyword queries on a set of documents. Each document consists of an arbitrary number of named fields, which are most commonly used to store the text of a document and its metadata.

In order to use Solr for the evaluation of KWQL queries, the metadata of wiki pages and further the metadata of its tags, fragments, and links are stored in a Solr document. The main principle of the translation is to materialize joins between content items and the directly connected resources.

The transformation of the resources connected to a content item to fields in the Solr index is lossy, since the value of multiple resources is stored in a single field. Thus, if multiple properties of a resource are queried, it cannot be guaranteed that hits in the index belong to the same resource.

To keep the index small, only dependencies to flat resources are materialized, which omits, in particular, nesting and linking of content items. Therefore, only queries that access content items together with their content, metadata, and directly related flat resources can be evaluated entirely in Solr. As soon as nesting and linking of content items come into play, however, we use Solr only to generate a set of candidates which match those parts of the query for which all necessary information is stored in the Solr index.

In order to evaluate a KWQL query through Solr, a portion of the KWQL query (that can be evaluated by Solr) is converted to the query language of Solr. Information which is not covered by the materialized joins and variables is either disregarded or at least converted to an existential quantification in order to reduce the number of false positives.

**7.8.1.2 Evaluation of Structural Constraints**

848

The second phase takes the structural parts of a query into account. All resources 849  
are represented as common objects in the KiWi system, and their dependencies are 850  
modeled by references between the interrelated objects. The objects are persisted 851  
using a common relational database in combination with an object-relational 852  
mapping. 853

In the current prototype, we validate the structural properties of a query for each 854  
candidate item individually. That means, nested resources (tags, fragments, links, 855  
and contained content items) which are specified in the query are considered by 856  
traversing the references of the currently investigated object. 857

We choose this approach, as structural constraints are often validated fairly 858  
quickly and far less selective than the keyword portions of KWQL queries. However, 859  
for future work, we envision an extension of KWilt that improves on the current 860  
implementation in two aspects: (a) It estimates whether the structural part is 861  
selective enough to warrant its execution without considering the candidates from 862  
the previous phase, followed by a join between the candidate sets from the two 863  
phases. (b) If structural constraints become more complex, specialized evaluation 864  
engines for hierarchical (XML-style) data, for example, a high-performance XPath 865  
engine, for link data, e.g., various graph reachability indexes, and for RDF data 866  
might be advantages. 867

In addition to the verification of the structural constraints, the structural dependencies 868  
of the contributing resources and the required values of their qualifiers are 869  
stored in relations which are needed during the last evaluation phase. 870

**7.8.1.3 Evaluation of First-Order Constraints over Wiki Resources**

871

In the final evaluation phase, first-order constraints over wiki resources are consid- 872  
ered, as induced by the KWQL variables (and some advanced features of KWQL 873  
such as injectivity). 874

Following constraint programming notation, we consider a first-order constraint 875  
a formula over logical relation on several variables. In order to use these constraints 876  
to express a KWQL query, every expression of a query that is involved in constraints 877  
not yet fully validated is represented by some variables. These variables are 878  
then connected using relations which reflect the structural constraints between 879  
the resources from the query and their metadata. These relations are constructed 880  
during the prior evaluation phase since all required values and dependencies of the 881  
resources are regarded in this phase anyhow. 882

Thus, the relations are used to connect the formal representation of the query and 883  
the candidate matches. The first-order constraints are evaluated using the constraint 884  
solver choco [21]. 885

Any content item that fulfills the constraints validated in all three phases is a 886  
match for the entire query. In fact, since we only feed candidate matches from 887

the prior phase to each subsequent phase, the content item (identifiers) returned by choco immediately gives us the KWQL answers.

888  
889

### 7.8.2 Skipping Evaluation Phases: KWQL's Sublanguages

890

The evaluation of a general KWQL query in KWilt is performed in three phases as described in the previous section. However, not all evaluation phases are required for every KWQL query. In the following, we give a characterization of KWQL queries that can be evaluated using only the first phase (and skipping the remaining ones), or only the first and second.

891  
892  
893  
894  
895

#### 7.8.2.1 Keyword KWQL or KWQL<sub>K</sub>

896

KWQL<sub>K</sub> is the restriction of KWQL to mostly flat queries where *resource terms* may not occur nested inside other resource terms and *structure terms* are not allowed at all.

897  
898  
899

Since tags and fragments itself cannot be nested more than one level, we can also materialize all tags and fragments for each content item. However, in contrast to (string-valued) qualifiers, a content item can have multiple tags or fragments. To allow evaluation with an information retrieval engine such as Solr, we have to ensure that multiple tag or fragment expressions always match with different tags or fragments of the surrounding content item. This avoids that we have to enforce the injectivity of these items in a later evaluation phase.

900  
901  
902  
903  
904  
905  
906

To ensure this, we allow tag and fragment queries but disallow

907

- Two keyword queries as sibling expressions in tag or fragment queries
- Two tag or fragment queries as sibling expressions

908  
909

KWQL<sub>K</sub> expressions can be evaluated entirely by the information retrieval engine, here Solr.

910  
911

#### 7.8.2.2 Tree-Shaped KWQL or KWQL<sub>T</sub>

912

KWQL<sub>T</sub> allows only queries corresponding to tree-shaped constraints, thus, no multiple occurrences of the same variable, and no potentially overlapping expression siblings.

913  
914  
915

We define an equivalence relation on expressions, called *potential overlap*, as a conservative approximation of overlapping. It holds between two expressions if they have the same return type in the KWQL semantics (see Section 7.5) or if the return type of one is a subset of that of the other one.

916  
917  
918  
919

KWQL<sub>T</sub> expressions can be evaluated by using only Solr and checking the remaining structural conditions in the second evaluation phase. Full first-order constraints are not needed, and the third (choco) phase can be skipped.

**Proposition 7.1.** *Given an arbitrary KWQL query, we can decide in linear time and space in the size of the query if that query is a KWQL<sub>K</sub> query and in quadratic time if it is a KWQL<sub>T</sub> query.*

*Proof.* From the definitions of KWQL<sub>K</sub> and KWQL<sub>T</sub>, it is easy to see that testing membership of a general KWQL expression can be done by a single traversal of the expression tree. In the case of KWQL<sub>T</sub>, we also have to test each (of the potentially quadratic) pair of siblings for overlap and store already visited variables. □

AQ15 The test whether a query is a KWQL<sub>T</sub> query can actually be achieved as a side effect of transforming the KWQL query into first-order constraints in the third evaluation phase constraints is generated; during this transformation, we need to execute the constraint solver at all. In practice, this is often cheaper than a separate test, as the generation of first-order constraints is fairly cheap and polynomial, except for queries with many potentially overlapping expression siblings.

### 7.8.3 Performance Evaluation

936

To analyze the performance of the KWilt prototype, the evaluation times of various queries of all three types were measured. For the experiment, the KiWi system was executed on a virtual server with a dual core 2.5 GHz processor and 4 GB of RAM running Ubuntu Linux.

In a first experiment, a number of queries, among them our example query from the introduction, were evaluated on a dataset consisting of 339 content items on the KiWi project. For all queries, preprocessing, that is, parsing, verification, and determining whether the query can be fully processed using only phase 1, was found to take between 27 and 42 ms. Table 7.8 further shows the processing times and number of results per query and processing phase. The first part of the table gives the numbers for queries that are covered by KWQL<sub>K</sub>. As the results show, these queries can overall be evaluated fairly quickly.

The second group of queries displayed in the table are those that can be evaluated using KWQL<sub>T</sub>. As the table illustrates, those queries can be evaluated quickly, but only if the first evaluation step has sufficiently reduced the candidate set. One underlying assumption behind KWilt is that most queries exclusively or predominantly use value-based selection criteria, that is, selection criteria that can be covered by the information retrieval engine in the first phase of the evaluation. When this assumption does not hold, the candidate set still contains a considerable amount of content items after the first evaluation phase. As the second evaluation phase is considerably slower than the first, evaluation times in such a situation can become very high. Correspondingly, the evaluation times for all four KWQL

**Table 7.8** Evaluation times in the KiWi dataset (339 content items)

Query	Phase 1		Phase 2		Phase 3		Total time [ms]
	Time [ms]	Results	Time [ms]	Results	Time [ms]	Results	
KiWi							t20.1
31	14	—	—	—	—	—	31
ci(text:KWQL title:KiWi)	6	1	—	—	—	—	6
KiWi tag(name:\$t)	42	9	—	—	—	—	42
ci(tag(name:KWQL) child:ci(tag(Example)))	10	5	44	1	—	—	54
ci(Munich link(target:ci(KiWi)))	33	4	52	4	—	—	85
ci(KiWi link(target:ci(KiWi)))	60	10	206	4	—	—	266
ci(tag(name:r)text :r)	149	9	53	9	103	9	305
ci(tag(author:admin) tag(name:KWQL))	9	5	55	5	67	4	131
ci(KiWi tag(name:\$t) link(target:ci(URI:\$u tag(name:\$t))))	44	9	181	4	194	3	419
							t20.10

queries are roughly inversely proportional to the size of the candidate set after the first evaluation phase (Table 7.9).

Finally, the lower three queries in the table make use of the full power of KWQL and require all three evaluation phases.

Overall, the results of this first experiment show that KWQL<sub>T</sub> queries can be evaluated using Solr with only little overhead for preprocessing the query. However, Solr queries involving wildcards are evaluated comparatively slowly. More critically, the second evaluation phase constitutes a bottleneck in the query evaluation process, particularly when the first phase does not sufficiently decrease the size of the candidate set.

In summary, this first small-scale evaluation of KWilt shows that the approach overall is viable and delivers good results as long as the underlying assumption holds true, namely, that most selection criteria used in queries are value based. As long as this is true, KWilt can employ Solr which quickly evaluates the query, either in total or by reducing the candidate set to a size that is manageable for the following evaluation phases.

**Table 7.9** Evaluation times in the RSS dataset (2049 content items)

Query		Phase 1		Phase 2		Phase 3		Total time [ms]
Time [ms]	Results	Time [ms]	Results	Time [ms]	Results			
tag(author:Mary)								t21.1
48	35	—	—	—	—	—	48	t21.2
semantic web								
51	22	—	—	—	—	—	51	t21.3
tag(name:web author:Peter)								
99	34	1769	34	—	—	—	1868	t21.4
ci(example title: <i>rtext :r</i> )								
105	38	21	38	15	1	141		t21.5
ci(title: <i>rtext :r</i> )								
679	505	15	505	75	58	769		t21.6
ci(tag(name:web) tag(author:Peter))								
92	34	863	34	3246	34	4201		t21.7

These, in particular the second evaluation phase, constitute the weak point of KWilt as it is currently implemented: the simple traversal of all candidate content items that constitutes the second phase in the current implementation performs very slowly. When a query does not use mainly value-based selection criteria or when the dataset is big, the size of the candidate set is not sufficiently decreased in the first evaluation phase and the second evaluation phase can take several seconds or longer.

Overall, the system delivers good results, but changes to the system are required to improve the performance of the second evaluation phase. The following section discusses possible steps that could be taken.

Despite the two possibilities for improving the second evaluation phase discussed above, namely, an evaluation strategy more closely tailored to the individual queries and their keyword and structure constraints and a reimplementation of the second evaluation phase using Web querying technology, two further changes could be employed to improve query performance:

- While saving all information about structurally connected content items in the index representation of a content item is clearly not practicable, some basic structural information could be represented. For example, the index could indicate whether a content item has any children or links to any other content items. Depending on how frequently nesting and linking relations are used in the wiki, this information could then help narrow down the candidate set, meaning that fewer content items have to be processed in the second evaluation phase.

- Queries that cannot be fully evaluated using Solr could be handled through a 997 translation into SQL that treats both the second and third evaluation phases. The 998 relational semantics given in Section 7.5 can serve as a basis for such a translation 999 of KWQL into SQL. The resulting alternative implementation of KWQL would 1000 not be based on the principle of gradually refining the query results like KWilt, 1001 but rather on choosing the best-suited tool before query evaluation begins. An 1002 evaluation of the resulting system could also show whether the use of Solr is 1003 justified, or whether translating fully translating KWQL into SQL is preferable. 1004

## 7.9 Outlook

1005

At least two extensions to KWQL as described here are desirable: to add two 1006 important features of keyword search to the language, fuzzy matching, and ranking 1007 should be provided. Toward this end, we suggest PEST [52], a PageRank-like 1008 approach to approximate querying of structured data that exploits the structure to 1009 propagate term weights between related data items and uses the resulting modified 1010 index for ranking as well as fuzzy matching over data structure. Secondly, one issue 1011 that has not been addressed so far is that of querying RDF with KWQL. While 1012 dealing with complex RDF graphs may indeed overburden many users, simple RDF 1013 triples are intuitive and easy to understand. KWQL should therefore allow users 1014 to query at least these simple RDF annotations that they and others have created. 1015 Weiand [51] discussed three solutions for adding support for RDF queries, one 1016 native and two based on the integration of existing RDF query languages. 1017

## 7.10 Conclusion

1018

The work presented in this chapter addresses the question how ease of use and 1019 rich functionality, two seemingly conflicting characteristics, can be consolidated in 1020 the context of the social semantic Web, and more specifically in the semantic wiki 1021 KiWi. We feel that this issue is crucial to the success of the social semantic Web: 1022 social semantic Web applications live from user participation and the adoption by 1023 a broad user base, but often fail to provide annotation and querying formalisms 1024 that allow casual and expert users alike to formalize knowledge and compose 1025 expressive queries to fully leverage the functionality of the application at hand. We 1026 presented KWQL, a query language for the KiWi wiki based on the label-keyword 1027 query paradigm that allows for rich combined queries of textual content, metadata, 1028 document structure, and annotations. 1029

We described the underlying principles and the syntax of KWQL, provided a 1030 formal semantics for the language, and discussed KWilt, an implementation of 1031 KWQL query evaluation based on a patchwork approach. We then distinguished 1032 three sublanguages of increasing complexity and showed that it is possible to 1033

efficiently recognize the sublanguage a given KWQL query belongs to and to adapt the evaluation process accordingly. The power of full first-order queries can be leveraged where needed, but at the same time, KWilt can evaluate basic queries at almost the speed of the underlying search engine, as we showed in a performance evaluation. Participants in a user study reacted positively to KWQL and visKWQL. They found the languages useful, expressive, and easy to use, at least given some time and practice. Even after a short introduction and a minimal amount of time to solve the assignments, participants overall were able to provide correct answers to more than half of the questions in a query writing task and over 80% of the questions in a query understanding task. 1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043

## References

1044

1. Auer, S., Dietzold, S., Lehmann, J., Riechert, T.: OntoWiki: a tool for social, semantic collaboration. Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge (2007) 1045  
1046  
1047
2. Aumueller, D.: Semantic authoring and retrieval within a wiki. Proceedings of the 2nd European Semantic Web Conference (2005a) 1048  
1049
3. Aumueller, D.: SHAWN: structure helps a wiki navigate. Proceedings of the BTW-Workshop WebDB Meets IR (2005b) 1050  
1051
4. Aumueller, D.: Towards a semantic wiki experience – desktop integration and interactivity in WikSAR. Proceedings of the 1st Workshop on the Semantic Desktop (2005c) 1052  
1053
5. Balmin, A., Hristidis, V., Koudas, N., Papakonstantinou, Y., Srivastava, D., Wang, T.: A system for keyword proximity search on XML databases. Proceedings of 29th International Conference on very Large Data Bases, pp. 1069–1072 (2003) 1054  
1055  
1056
6. Bao, J., Ding, L., Hendler, J.: Knowledge representation and query in semantic MediaWiki: a formal study. Technical Report TW-2008-42, Tetherless World Constellation (RPI) (2008) 1057  
1058
7. Bao, Z., Ling, T.W., Chen, B., Lu, J.: Effective XML keyword search with relevance oriented ranking. Proceedings of the 25th International Conference on Data Engineering, pp. 517–528 (2009) 1059  
1060  
1061
8. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword searching and browsing in databases using BANKS. Proceedings of the 18th International Conference on Data Engineering, pp. 431–440 (2002) 1062  
1063  
1064
9. Bischoff, K., Firan, C.S., Nejdl, W., Paiu, R.: Can all tags be used for search? Proceedings of the 17th ACM Conference on Information and Knowledge Management, pp. 193–202 (2008) 1065  
1066
10. Cohen, S., Mamou, J., Kanza, Y., Sagiv, Y.: XSearch: a semantic search engine for XML. Proceedings of 29th International Conference on very Large Data Bases, pp. 45–56 (2003) 1067  
1068
11. Dar, S., Entin, G., Geva, S., Palmon, E.: DTL's DataSpot: database exploration using plain language. Proceedings of 24rd International Conference on very Large Data Bases, pp. 645–649 (1998) 1069  
1070  
1071
12. El Ghali, A., Tifous, A., Buffa, M., Giboin, A., Dieng-Kuntz, R.: Using a semantic wiki in communities of practice. Proceedings of the 2nd International Workshop on Building Technology Enhanced Learning Solutions for Communities of Practice (2007) 1072  
1073  
1074
13. Fischer, J., Gantner, Z., Rendle, S., Stritt, M., Schmidt-Thieme, L.: Ideas and improvements for semantic wikis. Proceedings of the 3rd European Semantic Web Conference, pp. 650–663 (2006) 1075  
1076  
1077
14. Florescu, D., Kossmann, D., Manolescu, I.: Integrating keyword search into XML query processing. Comput. Networks **33**(1–6), 119–135 (2000) 1078  
1079

15. Fuchs, N.E., Kaljurand, K., Schneider, G.: Attempto Controlled English meets the challenges of knowledge representation, reasoning, interoperability and user interfaces. Proceedings of the 19th International Florida Artificial Intelligence Research Society Conference, pp. 664–669 (2006) 1080  
1081  
1082  
1083
16. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: XRANK: ranked keyword search over XML documents. Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 16–27 (2003) 1084  
1085  
1086
17. Haase, P., Herzig, D., Musen, M.A., Tran, T.: Semantic wiki search. Proceedings of the 6th European Semantic Web Conference, pp. 445–460 (2009) 1087  
1088
18. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**, 338–355 (1984) 1089  
1090
19. Hartl, A., Weiand, K., Bry, F.: visKQWL, a visual renderer for a semantic web query language. Proceedings of the 19th International Conference on World Wide Web, pp. 1253–1256 (2010) 1091  
1092
20. Hristidis, V., Papakonstantinou, Y., Balmin, A.: Keyword proximity search on XML graphs. Proceedings of the 19th International Conference on Data Engineering, pp. 367–378 (2003) 1093  
1094
21. Jussien, N., Prud'homme, C., Cambazard, H., Rochart, G., Laburthe, F.: choco: an open source java constraint programming library. Proceedings of the Workshop on Open-source Software for Integer and Constraint Programming (2008) 1095  
1096  
1097
22. Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H.: Bidirectional expansion for keyword search on graph databases. Proceedings of the 31st International Conference on very Large Data Bases, pp. 505–516 (2005) 1098  
1099  
1100
23. Kiesel, M.: Kaukolu: hub of the semantic corporate intranet. Proceedings of the 1st Workshop on Semantic Wikis (2006) 1101  
1102
24. Klein, B., Höcht, C., Decker, B.: Beyond capturing and maintaining software engineering knowledge – “Wikitologies” as shared semantics. Proceedings of the Workshop on Knowledge Engineering and Software Engineering (2005) 1103  
1104  
1105
25. Krötzsch, M., Vrandecic, D.: Semantic Wikipedia. In: Blumauer, A., Pellegrini, T. (eds.) Social Semantic Web, pp. 393–421. Springer, Berlin, Heidelberg, New York (2009) 1106  
1107
26. Kuhn, T.: AceWiki: a natural and expressive semantic wiki. CoRR abs/0807.4618 (2008) 1108
27. Ladwig, G., Tran, T.: Combining query translation with query answering for efficient keyword search. Proceedings of the 7th Extended Semantic Web Conference, pp. 288–303 (2010) 1109  
1110
28. Landefeld, R., Sack, H.: Collaborative web-publishing with a semantic wiki. Proceedings of the 1st Conference on Social Semantic Web, pp. 23–34 (2007) 1111  
1112
29. Leuf, B., Cunningham, W.: The Wiki Way: Quick Collaboration on the Web. Addison-Wesley, Reading, MA, USA (2001) 1113  
1114
30. Li, G., Feng, J., Wang, J., Zhou, L.: Effective keyword search for valuable LCAs over XML documents. Proceedings of the 16th ACM Conference on Information and Knowledge Management, pp. 31–40 (2007) 1115  
1116  
1117
31. Li, G., Feng, J., Wang, J., Song, X., Zhou, L.: SAILER: an effective search engine for unified retrieval of heterogeneous XML and web documents. Proceedings of the 17th International Conference on World Wide Web, pp. 1061–1062 (2008) 1118  
1119  
1120
32. Li, G., Ooi, B.C., Feng, J., Wang, J., Zhou, L.: EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 903–914 (2008) 1121  
1122  
1123
33. Li, J., Liu, C., Zhou, R.: XBridge: answering XML keyword search with structured queries (2008) 1124  
1125
34. Li, J., Liu, C., Zhou, R., Wang, W.: Suggestion of promising result types for XML keyword search. Proceedings of the 13th International Conference on Extending Database Technology, pp. 561–572 (2010) 1126  
1127  
1128
35. Li, Y., Yu, C., Jagadish, H.V.: Schema-free XQuery. Proceedings of the 13th International Conference on very Large Data Bases, pp. 72–83 (2004) 1129  
1130
36. Liu, Z., Chen, Y.: Identifying meaningful return information for XML keyword search. Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 329–340 (2007) 1131  
1132  
1133

37. Liu, Z., Walker, J., Chen, Y.: XSeek: a semantic XML search engine using keywords. Proceedings of the 33rd International Conference on very Large Data Bases, pp. 1330–1333 1134  
1135  
1136  
(2007)
38. Olteanu, D., Meuss, H., Furche, T., Bry, F.: XPath: looking forward. Proceedings of the EDBT Workshop on XML-based Data Management, pp. 109–127 (2002) 1137  
1138
39. Oren, E.: SemperWiki: a semantic personal wiki. Proceedings of the 1st Workshop on the Semantic Desktop (2005) 1139  
1140
40. Panagiotou, D., Mentzas, G.: A comparison of semantic wiki engines. Proceedings of the 22nd European Conference on Operational Research (2007) 1141  
1142
41. Qu, Y.: Q2RDF: ranked keyword query on RDF data. Technical Report, Southeast University, China (2008) 1143  
1144
42. Schaffert, S.: IkeWiki: a semantic wiki for collaborative knowledge management. Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, pp. 388–396 (2006) 1145  
1146  
1147
43. Schaffert, S., Bry, F., Baumeister, J., Kiesel, M.: Semantic wikis. IEEE Software **25**(4), 8–11 (2008) 1148  
1149
44. Schaffert, S., Eder, J., Grünwald, S., Kurz, T., Radulescu, M.: Kiwi – a platform for semantic social software. Proceedings of the 6th European Semantic Web Conference, pp. 888–892 1150  
1151  
1152 (2009)
45. Schmidt, A., Kersten, M.L., Windhouwer, M.: Querying XML documents made easy: nearest concept queries. Proceedings of the 17th International Conference on Data Engineering, pp. 321–329 (2001) 1153  
1154  
1155
46. Souzis, A.: Building a semantic wiki. IEEE Intel. Syst. **20**(5), 87–91 (2005) 1156
47. Tazzoli, R., Castagna, P., Campanini, S.: Towards a semantic wiki wiki web. Proceedings of the 3rd International Semantic Web Conference (2004) 1157  
1158
48. Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. Proceedings of the 25th International Conference on Data Engineering, pp. 405–416 (2009) 1159  
1160  
1161
49. Vagena, Z., Colby, L.S., Özcan, F., Balmin, A., Li, Q.: On the effectiveness of flexible querying heuristics for XML data. Proceedings of the 5th International XML Database Symposium, pp. 77–91 (2007) 1162  
1163  
1164
50. Wang, H., Zhang, K., Liu, Q., Tran, T., Yu, Y.: Q2Semantic: a lightweight keyword interface to semantic search. Proceedings of the 5th European Semantic Web Conference, pp. 584–598 1165  
1166  
1167 (2008)
51. Weiand, K.: Keyword-based querying for the social semantic web – the kwql language: concept, algorithm and system. PhD Thesis, University of Munich, Germany (2011) 1168  
1169
52. Weiand, K., Kneißl, F., Furche, T., Bry, F.: PEST: term-propagation over wiki-structures as eigenvector computation. Fifth workshop on semantic wikis (2010, in press) 1170  
1171
53. Xu, Y., Papakonstantinou, Y.: Efficient keyword search for smallest LCAs in XML databases. Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 537–538 (2005) 1172  
1173  
1174
54. Zhou, Q., Wang, C., Xiong, M., Wang, H., Yu, Y.: SPARK: adapting keyword query to semantic search. Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference, pp. 694–707 (2007) 1175  
1176  
1177

## AUTHOR QUERIES

- AQ1. Kindly check the corresponding author.
- AQ2. Please confirm the insertion of, “on the one hand,” in the sentence “As mentioned above,...”.
- AQ3. Please confirm the insertion of ‘his or’ in the sentence “The user can...”.
- AQ4. Kindly check the change made to the sentence starting “The most significant...”.
- AQ5. Please confirm the insertion of “On the one hand,” in the sentence “The different heuristics...”.
- AQ6. Please confirm the insertions of ‘he or’ in the sentence “Rather, the necessary...”.
- AQ7. Please confirm the insertion of ‘or she’ in the sentence “—what the user sees and perceives...”.
- AQ8. Please check if ‘and’ can be removed in the sentence “The KWQL semantics...”
- AQ9. Please confirm the insertion of apostrophe following ‘users’ in the sentence “As a consequence, ...”
- AQ10. Please confirm the changes in the sentence “... where participants were given a short...”
- AQ11. Kindly check whether the citations of Tables 7.7 and 7.9 are fine as cited.
- AQ12. Please check the sentence “These findings indicate that...”
- AQ13. ‘Performant’ in this para has been chaged as ‘Performance’. Please confirm
- AQ14. Please confirm the changes in the sentence “The three steps...”
- AQ15. Kindly rephrase the sentence starting “The test whether... “for better clarity
- AQ16. Kindly update Ref. [52].

# Chapter 8

## Semantic Link Discovery over Relational Data

Oktie Hassanzadeh, Anastasios Kementsietsidis, Lipyeow Lim,  
Renée J. Miller, and Min Wang

### 8.1 Introduction

From small research groups to large organizations, there has been tremendous effort in the last few years in publishing data online so that it is widely accessible to a large community. These efforts have been successful across a number of domains and have resulted in a proliferation of online sources. In the field of biology, there were 1,330 major online molecular databases at the beginning of 2011, which is more than a year earlier [14]. In the Linking Open Data (LOD) community project at the W3C, the number of published RDF triples has grown from 500 million in May 2007 to over 28 billion triples in March 2011 [29].

Fueling this data publishing explosion are tools for translating relational and semistructured data into RDF. Translation tools try to faithfully translate the

---

Part of this work has appeared in Proceedings of the 18th ACM Conference on Information and Knowledge Management [17]. ©2009 Association for Computing Machinery, Inc. Reprinted by permission.

AQ1

O. Hassanzadeh (✉) · R.J. Miller  
University of Toronto, Toronto, Ontario, M5S 3G4, Canada  
e-mail: [oktie@cs.toronto.edu](mailto:oktie@cs.toronto.edu); [miller@cs.toronto.edu](mailto:miller@cs.toronto.edu)

A. Kementsietsidis  
IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, USA  
e-mail: [akement@us.ibm.com](mailto:akement@us.ibm.com)

L. Lim  
University of Hawaii at Manoa, 1680 East West Road, POST 303E, Honolulu, HI 96822, USA  
e-mail: [lipyeow@hawaii.edu](mailto:lipyeow@hawaii.edu)

M. Wang  
HP Labs China, Beijing, China  
e-mail: [min.wang6@hp.com](mailto:min.wang6@hp.com)

structure and semantics of data. Relational data are often designed to ensure 16 the data remain consistent by minimizing redundancy. Warehouse data may be 17 designed to facilitate aggregation on specific dimensions of interest to a business. 18 Semistructured data are often designed to facilitate data exchange. No matter what 19 the *source* model, data translations typically result in data that are dirty (e.g., from 20 duplicate, incorrect, or invented values due to partial/missing mappings), and RDF 21 translation is no exception. To make matters worse, it is rare that organizational 22 data are designed to facilitate Web-scale data sharing. Hence, many of these data 23 sources, when published as RDF, may be missing internal semantics (links between 24 data values) that would assist in using them effectively as part of the Semantic Web. 25 And rarely do these data sources contain sufficient semantics to connect them with 26 other Semantic Web resources. 27

To enhance the usability of published data, we need to be able to create referential 28 links between data both within a data source and between different sources. 29 These links should have a known and specified semantics. Of course, discovering 30 such links requires the use of both approximate matching (to overcome syntactic 31 representational differences and errors) and semantic matching (to find specific 32 semantic relationships). These two types of matching must be used in concert to 33 accommodate the tremendous heterogeneity found in Web data and to accommodate 34 for errors or missing semantics within a data source. 35

In spite of their importance, research in discovering such semantic links has 36 mainly focused on a more restricted version of the problem, namely, on *entity* 37 *resolution* [11, 23], i.e., the identification of entities that represent the same *real-* 38 *world* entity. Techniques for finding more general semantic relationships may make 39 use of natural language semantics to improve the precision of information extraction 40 [1], but tend not to apply robustly to structured data on the Web. The importance of 41 discovering such links is also highlighted by the LOD project, where a number of 42 tools and frameworks have been developed that allow the generation and publication 43 of linked data from relational databases. Examples of such frameworks include 44 D2RQ [7], Triplify [3], and OpenLink's Virtuoso [12]. Although these frameworks 45 simplify the process of translating and publishing linked data, they do not provide a 46 link discovery mechanism so that data publishers can establish links to external 47 sources (or find implicit internal links). For typical users, this means they must 48 manually experiment with a myriad of different link discovery methods to find one 49 that suits their needs. 50

In this chapter, we present LinQuer, a generic and extensible framework for 51 integrating link discovery methods over relational data. The goal of this framework 52 is to facilitate experimentation and help users find and combine the link discovery 53 methods that will work best for their application domain. To ease experimentation, 54 the framework is declarative and permits the interleaving of standard data manipu- 55 lation operators with link discovery. 56

The LinQuer framework permits the discovery of links within and between 57 relational sources using LinQL, an extension of SQL that integrates querying with 58 link discovery. LinQL includes a variety of native link discovery methods and is 59 extensible to additional methods, written in SQL or as user-defined functions (UDF). 60

This permits users to interleave declarative queries with interesting combinations of link discovery requests. The link discovery methods may be syntactic (approximate match or similarity functions), semantic (using ontologies or dictionaries to find specific semantic relationships), or a combination of both.

In this chapter, we show that by integrating ad hoc querying and a rich collection of link discovery methods, the LinQuer framework supports rapid prototyping, testing and comparison of link discovery methods. A common way to use this framework would be to declaratively specify a portion of the data of interest (over which accuracy can be assessed) and to invoke one or more link discovery methods. The accuracy of the results can be evaluated by a user or an automated technique, and the specification of the link method can be interactively refined to produce better results.

Often, link discovery algorithms are implemented using general programming languages by third-party developers, and are automatically invoked with arguments defined through the declarative specification. For data publishers, these programs act as *black-boxes* that sit outside the data publishing framework and whose modification requires the help of their developers. The LinQuer framework addresses these shortcomings by leveraging native SQL implementations for a number of link discovery algorithms. This approach has several advantages: (a) this framework can be easily implemented on existing relational data sources with minimum effort and without any need for externally written program code; (b) we can take advantage of the underlying DBMS query engine optimizations while evaluating the SQL implementations of the link methods; and (c) we can support efficiency and functionality enhancements (e.g., a link index) to improve the efficiency and accuracy of linking algorithms.

We also show how the declarative invocation of link methods permits users to tune link methods and their performance. The native support for methods permits customization where domain knowledge is available. We give examples where domain knowledge can be specified in the database and used to greatly enhance the performance of the discovery process. Finally, we describe a case study of how this framework can be used to discover links over real clinical trial data drawn from a number of disparate Web sources.

The rest of the chapter is organized as follows. Section 8.2 introduces our running example, while Sect. 8.3 describes how links between data sources can be specified declaratively and Sect. 8.4 describes the linkage methods that LinQuer provides natively. Section 8.5 presents the algorithms for translating the link specifications into SQL queries. Our experimental study is described in Sect. 8.6. Section 8.7 highlights related work and we conclude in Sect. 8.8.

## 8.2 Motivating Example

In this chapter (and in our case study in Sect. 8.6), we use an example from the health care domain drawn from a set of real-world data sources. One such source

AQ2 is a clinical trials' database which includes the sample relation in Fig. 8.1a. For each trial, the *CT* relation stores its identifier *trialid*, the *condition* considered, the suggested *intervention*, as well as the *location*, *city*, and a related *publication*. Another source stores patient electronic medical records (EMR) and includes a patient visit relation *PV* (Fig. 8.1b), which stores for each patient visit its identifier *visitid*, the *diagnosis*, recommended *prescription*, and *location*. Finally, we consider a Web source extracted from DBpedia [6], which stores information about drugs and diseases and includes the *DBPD* and *DBPG* relations (Fig. 8.1c, d) that store the *name* in DBpedia of diseases and drugs, respectively.

We now describe briefly some types of links users and data publishers may like to discover between these sources. For the *CT* and *PV* relations, we note that the *condition* column in the *CT* relation is semantically related and can be linked to the *diagnosis* column in the *PV* relation. Such links may be useful to clinicians since they associate a patient's condition with related clinical trials, and might be used to suggest alternative drugs or interventions. In Fig. 8.1, patient visit "VID770" with diagnosis "Thalassaemia" in the *PV* relation may be linked to the trial "NCT00579111" with condition "Hematologic Diseases" since "Thalassaemia" is a different representation of "Thalassemia" and according to the NCI medical thesaurus [24] "Thalassemia" is a type of "Hematologic Diseases." As this example illustrates, a clinician may be interested in not only *same-as* relationships, but also hyponym relationships such as *type-of*. Similarly, note that the *intervention* column in the *CT* relation can be linked to the *prescription* column in the *PV* relation. Such links can provide evidence for the relevance and effectiveness of a drug for a particular condition. For example, both patient visits in Fig. 8.1b may link to trial "NCT00336362" (Fig. 8.1a) based on the fact that "hydroxycarbamide," "Hydroxyura," and "Hydroxyurea" all refer to the same drug.

Additional links are possible if one considers the existence of links between the locations of patients and the presence of clinical trials in these locations. As an example, "Westchester Med. Ctr" from visit "VID777" could link to "Columbia University" based on geographic proximity. Another interesting link discovery scenario arises when a user who is interested in a particular trial, wants to find other related trials based on certain criteria, for example, the similarity of the title and

<i>trialid</i>	<i>cond</i>	<i>inter</i>	<i>loc</i>	<i>city</i>	<i>pub</i>
NCT00336362	Beta-Thalassemia	Hydroxyurea	Columbia University	New York	14988152
NCT00579111	Hematologic Diseases	Campath	Texas Children's Hospital	Austin	3058228

(a) Clinical trials (*CT*)

<i>visitid</i>	<i>diag</i>	<i>prescr</i>	<i>location</i>
VID770	Thalassaemia	Hydroxyura	Texas Hospital
VID777	PCV	Hydroxycarbamide	Westchester Med. Ctr

(b) Patient visit (*PV*)

<i>name</i>
Thalassemia
Blood_Disorders

(c) DBpedia Disease (*DBPD*)

<i>name</i>
Alemtuzumab
Hydroxyurea

(d) DBpedia Drug (*DBPG*)**Fig. 8.1** Sample relations

authors of the trials' corresponding publications. Obviously, to be effective, links 134  
should be tolerant of errors and differences in the data, such as typos or abbreviation 135  
differences. 136

Data publishers often build online Web-accessible views of their data. In such 137  
settings, they often want to provide links between their data and those in other online 138  
Web sources. As an example, a Web source of clinical trials requires links to other 139  
Web sources related to the trials like, say, the DBpedia or YAGO [25] sources. In our 140  
sample relations, the above example translates into finding links between the *cond* 141  
and *inter* columns of *CT* and the *name* column of the *DBPD* and *DBPG* relations, 142  
respectively. The online trials data source can link the condition “Hematologic 143  
Diseases” to DBpedia resource (or Wikipedia page) on “Blood\_Disorders,” and 144  
link the intervention “Campath” to DBpedia resource “Alemtuzumab” using the 145  
semantic knowledge that “Campath” is the brand name for the chemical name 146  
“Alemtuzumab.” In our work, we will focus on the implementation, evaluation, and 147  
testing of link discovery methods. We will be agnostic as to how the discovered 148  
links are published. A user can create his or her own identifiers for specific semantic 149  
relationship (e.g., *brandNameOf*) or reuse identifiers for common relationships (e.g., 150  
*owl:sameAs*). 151

### 8.3 The LinQL Language

152

In this section, we introduce the LinQL language. LinQL extends SQL with constructs 153  
that allow declarative specification of methods and requirements for linking values 154  
or records in an RDBMS. We show several features of LinQL, how it can be used to 155  
define linkage methods, how the methods can be used inside an SQL statement for 156  
link discovery, and how indices can be defined to speed up linkage discovery. 157

Figure 8.2 shows the main constructs of the LinQL grammar (the full grammar can 158  
be found elsewhere [30]). A *link specification*, or *linkspec* for short, defines the con- 159  
ditions that two given values must satisfy before a link can be established between 160  
them. As shown in Fig. 8.2, a `CREATE LINKSPEC` statement defines a new linkspec 161  
and accepts as parameters the name of the linkspec, the linkage method, and its 162  
parameters. LinQL provides several *native* (or *built-in*) linkage methods including 163  
semantic (synonym and hyponym), and a variety of syntactic (string matching) 164  
similarity measures. See more details on the native methods in the following section. 165  
Native methods can be used as such, or they can be customized through parameters. 166

*Example 8.1.* A common string similarity measure that has been shown to have 190  
good accuracy and efficiency is the token-based weighted Jaccard measure [16]. 191  
Like all our methods based on similarity, this method takes a similarity threshold  $\theta$  192  
as input. Only pairs of values with similarity over  $\theta$  will be output as a link. The 193  
weighted Jaccard is a measure of set similarity over the set of tokens in two strings. 194  
A token is either a substring of length  $q$  (an input parameter to the method) or a 195  
word (when  $q = 0$ ). This measure is supported by LinQL as a native linkage method 196

```

linkspec_stmt := CREATE LINKSPEC linkspec_name          167
                  AS link_method opt_limit;           168
169
linkindex_stmt := CREATE LINKINDEX opt_idx_args        170
                  linkindex_name ON table(col)           171
                  USING native_method;                 172
173
link_method := native_method | link_clause_expr | UDF; 174
175
native_method := (synonym | hyponym | string_method) opt_args; 176
177
string_method := jaccard | weightedJaccard | cosine | bm25 | hmm | ...; 178
179
link_clause_expr := link_clause AND link_clause_expr    180
                   | link_clause OR link_clause_expr   181
                   | link_clause;                      182
183
link_clause := LINK source WITH target                184
                  USING link_terminal opt_limit;       185
186
link_terminal := native_method | UDF opt_args | linkspec_name; 187
188
opt_limit := LINKLIMIT number;                      189

```

**Fig. 8.2** The LinQL grammar (Version 1.1)

called *weightedJaccard*. A user can create a link specification using this measure 197  
 by setting the parameters used in the similarity computation. For example, he or she 198  
 may set the threshold parameter to 0.5, tokenize using q-grams of size 2, and set the 199  
 maximum string length to 50, creating the following link specification: 200

```

CREATE LINKSPEC myJaccard1                         201
AS weightedJaccard (0.5, 2, 50);                  202

```

Now this link specification can be used as a join predicate in queries by any user. 203  
 Notice that this specification does not indicate processing constraints. 204

A link specification can also be defined in terms of *link clause expressions*. These 205  
 are Boolean combinations of link clauses, where each link clause is semantically a 206  
 Boolean condition on two columns and is specified using either (a) a native method; 207  
 (b) a user-defined function (UDF); or (c) a previously defined linkspec. 208

*Example 8.2.* Consider a setting in which a link between two values is established if 209  
 a semantic relationship (e.g., synonymy or hyponymy) exists between these values 210  
 in an ontology. This scenario commonly occurs in a number of domains, including 211  
 health care, where sources are free to use their own local vocabularies (e.g., diag- 212  
 nosis and drug names) as long as these vocabularies can eventually be matched 213

through a commonly accepted ontology (e.g., the NCI thesaurus [24]). Assume that 214  
the ontology is stored in table *ont* with concept IDs in column *cid* and terms in *term*. 215

The following linkspec illustrates the power of link clauses. It creates a link 216  
between two values *src* and *tgt* if their corresponding terms in an ontology are 217  
synonyms of each other. In the linkspec, the *weightedJaccard* native method 218  
(customized as in the previous example) is used to match the values to corresponding 219  
terms in the ontology (accounting this way for possible syntactic errors in the 220  
values). Then, the corresponding terms are tested for synonymy through the 221  
*synonym* native method. 222

```
CREATE LINKSPEC mixmatch
AS LINK src WITH tgt
  USING synonym(ont,cid,term) LINKLIMIT 10
  AND
  LINK src WITH ont.term
  USING myJaccard1 LINKLIMIT 10
  AND
  LINK ont.term WITH tgt
  USING myJaccard2 LINKLIMIT 10; 231
```

Clearly, semantic links are not necessarily one-to-one and, in general, a value 232  
from one relation can be linked to more than one value from a second relation. For 233  
example, it is common for drugs to have more than one name. Therefore, while a 234  
drug appears as “aspirin” in one relation it might appear as “acetylsalicylic acid” 235  
or “ASA” in another. When multiple such links are possible, users may (optionally) 236  
limit the number of such links and only consider *k* results, or the *top-k* where order- 237  
ing is possible. The *LINKLIMIT* essentially specifies the value of this *k* parameter. 238

*Example 8.3.* In the previous example, while defining the *mixmatch* linkspec, 239  
*LINKLIMIT* is set equal to 10, for all three link clauses. Hence, only the top 10 240  
links are considered in each method. 241

The previous examples consider the *local* version of the *LINKLIMIT* construct 242  
that is associated with a particular clause. The LinQL grammar also includes a 243  
*global* *LINKLIMIT* construct that is associated with the whole *CREATE LINKSPEC* 244  
statement. This can be thought of as a postprocessing filter of the links returned by 245  
the linkspec methods used in the statement. 246

Another feature of LinQL is supporting Boolean valued user-defined functions 247  
(UDFs). In the next section, we discuss in detail the differences and advantages 248  
of native linkage methods over UDFs. Here, we just mention that a distinguishing 249  
characteristic of the two is that the former are implemented in SQL, while the latter 250  
are not. UDFs essentially provide (desirable) extensibility to the LinQuer framework 251  
since the framework can incorporate newly or existing linking methods that either 252  
have not been implemented yet in SQL or cannot be implemented in SQL at all. 253

AQ4

*Example 8.4.* Suppose a user writes a UDF that implements his or her own similar- 254  
ity function. This UDF, *myLinkUDF*(*thr*, *delC*, *insC*, *subC*), returns true only 255

if the value of edit similarity (a popular string similarity measure [11]) of the values 256  
on which it is applied is above the threshold value `thr` (where `delC`, `insC`, and 257  
`subC` are the costs of delete, insert, and substitute operations, respectively). Then, 258  
the following linkspec can use that UDF as follows: 259

```
CREATE LINKSPEC myLink
AS myLinkUDF(0.5, 1, 1, 1);
```

So far, we have only looked at how linkspecs are defined. We now show how 260  
linkspecs are used inside queries. 261

*Example 8.5.* In the sample relations of Fig. 8.1, suppose we want to find links 264  
between the diagnosis of patient visit with id 1234 and the condition of a trial in 265  
the clinical trials relation *CT* that is located in New York city, using the native 266  
weightedJaccard method with default parameter values as the linkspec. Then, the 267  
following query can be used: 268

```
SELECT PV.*, CT.*
FROM visit PV, trial CT
WHERE PV.visitid = 1234 AND
      CT.city='New York' AND
      LINK PV.diag WITH CT.cond
      USING weightedJaccard LINKLIMIT 10;
```

Notice that the linkspec here is essentially defined *inline*. For more complex 275  
linkspecs, or for situations where the same linkspec is used multiple times by one or 276  
more users, the query can refer to a previously defined linkspec in a similar fashion. 277  
This is a way for a DBA to provide a set of specifications for methods using the *best* 278  
parameter settings for different domains, making these methods more accessible to 279  
less expert users who may not know how to set the parameters. For example, in the 280  
query above, we can use the mixmatch linkspec instead of the weightedJaccard, as 281  
follows: 282

```
SELECT PV.*, CT.*
FROM visit PV, trial CT
WHERE PV.visitid = 1234 AND
      CT.city='New York' AND
      LINK PV.diag WITH CT.cond
      USING mixmatch LINKLIMIT 10;
```

For cases where a linkspec is used many times over a possibly large set of records, 289  
LinQL allows definition of an index (or linkindex) over columns for specific linkage 290  
methods. These indices can significantly lower query execution times. 291

*Example 8.6.* Consider the previous example, and assume that we would like to 292  
 improve the performance of the weightedJaccard-based linking. We can do that 293  
 by issuing the following LINKINDEX statements which create indices appropriate for 294  
 the particular linking method: 295

```
CREATE LINKINDEX FOR visit.diag USING weightedJaccard; 296
CREATE LINKINDEX FOR trial.cond USING weightedJaccard; 297
```

## 8.4 Native Linkage Methods

298

In what follows, we discuss the generic link finding primitives that are *necessary* 299  
 to effectively discover links in *real-world scenarios*. We show that LinQL not only 300  
 can accommodate these primitives, but also supports them in the form of *native* 301  
 linkage methods. We show how these native methods are translated into standard 302  
 SQL queries in the next section. 303

### 8.4.1 Similarity Functions as Linkage Methods

304

Over many domains, there is a natural notion of similarity. Any similarity function 305  
 $sim(v_1, v_2)$  that quantifies how close two values within a domain are can form the 306  
 basis of a linkage method. Some similarity functions are asymmetric, in which case 307  
 we will refer to the first argument as the *base* and the second as the *target*. Any 308  
 similarity function together with a similarity threshold  $\theta$  can be used as a linkage 309  
 method. In such a method, there is a link between two input values if their similarity 310  
 score, returned by function  $sim()$ , is above  $\theta$ . The right value of the threshold 311  
 depends on the characteristics of the dataset, the similarity function, and the applica- 312  
 tion. The user can find a good value for the threshold for each application by trying 313  
 different thresholds and evaluating the accuracy of (a subset of) the links that result. 314

### 8.4.2 String Similarity Functions

315

In publishing data on the Web, string data have a special prominence. String data 316  
 are prone to several types of inconsistencies and errors including typos, spelling 317  
 mistakes, use of abbreviations or different conventions. Therefore, finding similar 318  
 strings, often referred to as approximate (or fuzzy) string matching (or approximate 319  
 join) [8], is an important feature of a link discovery framework. 320

There exist a variety of similarity functions for string data in the literature [8]. 321  
 The performance of a similarity function usually depends on the characteristics of 322

data, such as length of the strings, and the type errors and inconsistencies present in the data. As stated earlier, in LinQuer we are interested in algorithms that are fully expressible in SQL (the benefits of which are well known [15, 16]). There are additional benefits of this choice for this framework. Specifically, the use of SQL as an implementation language for linkage methods permits on-the-fly calculations of the similarity scores that can be enhanced dynamically to increase the functionality of the matching algorithm by relying on characteristics of the domain of the source and target relations. Moreover, custom indexing techniques can be used to speed up the query execution time.

A popular class of string similarity functions is based on tokenization of the strings into q-grams, i.e., substrings of length  $q$  of the strings. By using q-gram tokens, we can treat strings as sets of tokens, and use a set similarity measure as the measure of similarity between the two strings. Furthermore, q-gram generation, storage and set similarity computation can all be done in SQL. This makes the following class of functions suitable for this framework:

- Size of the *intersection* of the two sets of q-grams, i.e., the number of common q-grams in the two strings.
- *Jaccard similarity* of the two sets of q-grams which is the size of the intersection divided by the size of the union, i.e., the percentage of common q-gram tokens between the two strings.
- Weighted version of the above measures. The weight of each q-gram is associated with its *commonality* in the base (or target or both) data sources. The higher the weight of a q-gram, the more important the q-gram is. For example, when matching diagnosis across medical sources, q-grams for commonly occurring strings like “Carcinoma” or “Cancer” should have low weights so that the value of the similarity function for the strings “Renal cell carcinoma” and “Squamous cell carcinoma” is small, compared to that for the strings “Renal cell carcinoma” and “Renal cell cancer.”

The other string similarity measures suitable for this framework include methods derived from relevance functions developed for documents in information retrieval, namely Cosine with tf-idf, Okapi-BM25, Language Modeling and Hidden Markov Models [16]. There are several other string similarity measures including but not limited to edit similarity, Jaro, Jaro-Winkler, SoftTFIDF, and Generalized Edit similarity [9, 16]. These functions can be implemented using a UDF. In this chapter, we only focus on the above q-gram-based set-similarity measures and refer the readers to the project’s online documentation page for a list of all supported linkage methods [30].

#### 8.4.2.1 Token Weight Assignment

To assign weights to q-gram tokens, we use an approach inspired by the Inverse Document Frequency (IDF) metric in information retrieval. IDF weights reflect the commonality of tokens in documents with tokens that occur more frequently

in the documents having less weight. So, by analyzing (offline) the q-gram token frequency, we assign less weight to common tokens like “Disorder” or “Cancer” in a medical source. As a functionality enhancement, we also let the user manually specify or adjust the weights of some tokens in a user-defined table. These weights override the automatically assigned (IDF-based) weights for these tokens. Manual weight adjustment is useful in applications where the user has prior knowledge about the importance of some tokens. For example, when matching diagnosis across sources, the user knows that often the use of numbers plays a more important role in the diagnosis than the name of the disease itself. So, by assigning a very low (or negative) weight to numbers, wrong matches between highly similar strings like “Type 1 Diabetes” and “Type 2 Diabetes” can be avoided. Similarly, when matching conditions (e.g., “Diabetes”, “Cancer”) from an online source such as WebMD to their corresponding entries in, say, Wikipedia, the conditions in Wikipedia might include the term “(disease)” to disambiguate the disease from other terms with the same name (e.g., “Choreia” is a medical disorder, but also an ancient Greek dance), although “disease” may not occur very frequently over all disease names (since most disease names are unique). Knowing this, the user can adjust the weight of the token “disease” to increase the likelihood of a correct link.

#### 8.4.3 Semantic Matching Methods

382

Link discovery between values often requires the use of domain knowledge. In a number of domains, there are existing, commonly accepted, semantic knowledge bases that can be used to this end. In domains where such semantic knowledge is not available, users often manually define and maintain their own knowledge bases.

A common type of such semantic knowledge is an ontology. In the health care domain, well-known ontologies such as the NCI thesaurus [24] are widely used and encapsulate a number of diverse relationship types between their recorded medical terms, including, synonymy, hyponymy/hypernymy, etc. Such relationship types can be conveniently represented in the relational model and (recursive) SQL queries can be used to test whether two values are associated with a relationship of a certain type [21]. Therefore, semantic knowledge in the form of ontologies can be seamlessly incorporated in LinQuer and used for the discovery of links. So, while considering links between two sources, semantic knowledge can be used to link a diagnosis on “Pineoblastoma” to one on “PNET of the Pineal Gland,” since the two terms are synonyms of each other. Similarly, a diagnosis on “Brain Neoplasm” can be potentially linked with both of the previous diagnoses, since the latter term is a hypernym of the former terms. Semantic matching is an important complement to string matching and both can enhance the recall of link discovery.

400

#### 8.4.4 Indexing for Native Linkage Methods

401

As mentioned earlier, the main goal of the LinQuer framework is to provide genericity, extensibility, and ease of use for linkage methods. To achieve scalability without sacrificing these goals, LinQuer provides a set of indexing techniques for native linkage methods within the framework that depending on the linkage method and application can be used during query evaluation to improve performance. These techniques rely on preprocessing strategies based on the specific linkage methods that will be used on specific columns. A basic strategy is materialization and indexing of parts of the similarity score calculation algorithms to avoid repetition. For string matching, additional indexing and hashing techniques can be used for the above-mentioned string similarity predicates. Examples of such techniques include the all-pairs indexing algorithms of Bayardo et al. [4], the *Weighted Enumeration* (WTENUM) signature generation algorithm [2], and *Locality Sensitive Hashing* [20].

414

### 8.5 From LinQL to SQL

415

In what follows, we describe the main algorithm for translating a LinQL query to an SQL query, and then describe the algorithm for implementing each native linkage method. At the end, we discuss the effect of `LINKINDEX` definitions on the implementation of the linkage methods.

419

Algorithm `LINQL2SQL` translates a single LinQL query to an SQL query (or script) by first splitting the LinQL query into a base query  $Q_{base}$  (which is in SQL) and a link clause expression which is a Boolean expression in disjunctive normal form (DNF) over a set of link clauses. The algorithm `LINKCLAUSEEXPR2SQL` considers each disjunction in the link expression creating an SQL query for each and then returns the SQL `UNION ALL` of these queries. Within a disjunct, the link clauses are applied in order by calling `LINKCLAUSEEXPR2SQL` for each conjunct. Note that the rewritings applied by `LINKCLAUSEEXPR2SQL` are specific to the type of link clause used (native or UDF) and these rewriting may not be commutative. In Example 8.2, instead of applying the `mixmatch` linkspec, a developer could have directly used an approximate string link specification (via `weightedJaccard` or another method) and then applied a semantic ontology linking. This may yield different results than a query that applies the ontology linking first followed by approximate string linking over the ontology terms. In the former case, data terms must match the ontology exactly, in the latter case the linking with the ontology will be tolerant of string errors.

435

Note that we assume that before running the LinQL query statement, the `linkspe` and `linkindex` statements are processed and loaded into a system definitions repository. `LINKCLAUSE2SQL` parses a link clause to determine what type of link terminal is used. If the link terminal is a UDF, we simply add an invocation of the

439

UDF (and its parameters if any) in the where clause of the SQL base query. If the link terminal is a native method, we rewrite the SQL base query using the rewrite rules associated with that particular native method. If the link terminal is a reference to a named linkspec, we retrieve the associated linkspec statement from the system definitions repository and parse the associated link method. The link method can be a UDF, native method or link clause expression. UDFs and native methods are translated as described previously. Link clause expressions are translated by a recursive call to the LINKCLAUSEEXPR2SQL sub-routine. The recursion stops when either a UDF or a native method is encountered.

The main translation logic is in the rewriting rules associated with the native methods. A native method's rewriting rules are specified in two parts: view definitions and link conditions. For example, the rewriting rules for the weightedJaccard native method on column  $col1$  of table  $tab1$  and column  $col2$  of table  $tab2$  consist of the view definitions  $tab1col1tokens$ ,  $tab1col1tokenweights$ ,  $tab1col1weights$ ,  $tab1col1sumweights$ ,  $tab2col2tokens$  that are needed for similarity score calculation, view definition  $scores$  for score calculation, and the link conditions  $scores.col1=tab1.col1 \text{ AND } scores.col2=tab2.col2$  and additional conditions depending on the similarity threshold, the value of  $\text{LINKLIMIT}$  and those given by the user. The use of the view definition syntax here is purely for readability. In practice, the SQL queries associated with the view definitions can be inlined into the actual query itself (resulting in a possibly hard to read query). The WITH statement supported by some DBMS (including the DMBS we used, IBM DB2) is another means for inlining some of the view definitions. Depending on the application, one

---

**Algorithm 15:** The LINQL2SQL Algorithm
 

---

**input :** LinQL query  $L$   
**output:** SQL query  $Q$

$lce \leftarrow$  extract link clause expression in DNF from  $L$ ;  
 $Q_{base} \leftarrow L - lce$ ;  
 $Q \leftarrow \text{LINKCLAUSEEXPR2SQL}( Q_{base}, lce )$ ;  
**return**  $Q$ ;

---



---

**Algorithm 16:** The LINKCLAUSEEXPR2SQL Algorithm
 

---

**input :** An SQL base query  $Q_{base}$ , a link clause expression  $lce$   
**output:** SQL query  $Q$

$i \leftarrow 0$ ;  
**forall**  $disjuncts D$  in  $lce$  **do**  
 |  $Q_i \leftarrow Q_{base}$  ;  
 | **foreach** next link clause  $l$  in  $D$  **do**  
 | |  $Q_i \leftarrow \text{LINKCLAUSE2SQL}( Q_i, l )$ ;  
 | |  $i \leftarrow i + 1$ ;  
 % final query is the SQL 'UNION ALL' of all  $Q_i$  ;  
 $Q \leftarrow \cup_i Q_i$  ;  
**return**  $Q$ ;

---

**Algorithm 17:** The LINKCLAUSE2SQL Algorithm

---

```

input : A SQL base query  $Q_{base}$ , a link clause  $l$ 
output: SQL query  $Q$ 

 $Q \leftarrow Q_{base} ;$ 
if  $link\_terminal(l) = UDF$  then
| add UDF invocation to where clause in  $Q$ ;
else if  $link\_terminal(l) = native\_method$  then
| rewrite  $Q$  using native_method's rewriting rules;
else if  $link\_terminal(l) = linkspec\_name$  then
| get link_method from associated linkspec_stmt in system definitions repository;
| if  $link\_method = UDF$  then
| | add UDF invocation (with the optional parameters) to where clause in  $Q$ ;
| else if  $link\_method = native\_method$  then
| | rewrite  $Q$  using native_method's rewriting rules;
| else if  $link\_method = link\_clause\_expr$  then
| |  $lce \leftarrow$  get associated link_clause_expr;
| |  $Q \leftarrow \text{LINKCLAUSEEXPR2SQL}(Q_{base}, lce);$ 

return  $Q;$ 

```

---

may choose to materialize all or part of these views using `LINKINDEX` statements 463  
in order to speed up the query time. The rest of this section describes the rewriting 464  
rules used to implement some of the native link methods. We first present the SQL 465  
queries based on view definitions and then discuss the effect of `linkindex` definitions. 466

### 8.5.1 Approximate String Matching Implementation

467

The SQL rewriting of the native link specification for approximate string matching 468  
consists of three steps, namely, (1) the (creation of views for) tokenization of strings 469  
into q-grams or word tokens; (2) the gathering of statistics and calculation of token 470  
weights; and (3) the calculation of link scores based on the weights. In more detail: 471

Step 1: This step can be done fully in SQL using standard string functions present 472  
in almost every DBMS, or with the addition of small (efficient) UDFs. Assume 473  
a table `integers` exists that stores integers 1 to  $N$  (maximum allowable length 474  
of a string). The main idea is to use basic string functions `SUBSTR` and `LENGTH` 475  
along with the sequence of integers in table `integers` to create substrings of length 476  
 $q$  from the strings in column `col1` of table `table1` with primary key `tid1`. The 477  
following SQL code shows this idea for  $q = 3$ : 478

```

SELECT tid1 as tid, SUBSTR(col1,integers.i,3) as token
FROM   integers INNER JOIN table1
        ON integers.i <= LENGTH(col1) - 2

```

479

480

481

In practice, the string in *coll* is used along with `UPPER()` (or `LOWER()`) functions 482 to make the search case insensitive. Also, the string is padded with  $q - 1$  occurrences 483 of a special character not in any word (e.g., “\$”) at the beginning and end using the 484 `CONCAT()` function. Similarly, the spaces in the string are replaced by  $q - 1$  special 485 characters. In case of tokenization using word tokens a similar SQL-based approach 486 can be used. At the end of this process, the token generation queries are declared as 487 views to be used in the following steps. 488

*Steps 2 and 3:* These steps are partly native-link specific, so we present them 489 through an example. In what follows, we use the `weightedJaccard` native method 490 as an example. 491

*Example 8.7 (weightedJaccard native method).* The following LinQL query 492 considers patient visit 1234 and finds (through `weightedJaccard`) the clinical trial 493 records in New York City that match the visit condition: 494

```
SELECT PV.visitid, CT.trialid
FROM visit AS PV, trial AS CT
WHERE PV.visitid = 1234 AND CT.city='NEW YORK' AND
      LINK PV.diag WITH CT.cond
      USING weightedJaccard
```

This specification is translated into the following SQL queries. Initially, three 500 queries calculate the IDF weights for the tokens and the auxiliary views needed for 501 the final score calculation: 502

```
CREATE VIEW visit_diagnosis_tokenweights AS
SELECT token, LOG(size - df + 0.5) - LOG(df+0.5) as weight
FROM   ( SELECT token, count(*) as df
        FROM   (SELECT * FROM visit_diagnosis_tokens
                GROUP BY tid, token) f
        GROUP BY token ) D,
        ( SELECT count(*) as size
          FROM visit_diagnosis_tokens ) S
CREATE VIEW visit_diagnosis_weights AS
SELECT tid, B.token, weight
FROM visit_diagnosis_tokenweights idf,
     (SELECT DISTINCT tid, token
      FROM visit_diagnosis_tokens B) B
WHERE B.token = idf.token
CREATE VIEW visit_diagnosis_sumweights AS
SELECT tid, sum(weight) as sumw
FROM visit_diagnosis_weights
GROUP BY tid
```

Then, the next query returns the links along with their final scores:

523

```

WITH      scores(tid1, tid2, score) AS (          524
    SELECT tid1, tid2,
           (SI.sinter / (BSUMW.sumw+QSUMW.sumw-SI.sinter)) 525
           AS score
  FROM      (SELECT BTW.tid AS tid1, QT.tid AS tid2,      526
              SUM(BTW.weight) AS sinter
            FROM      (SELECT * FROM visit_diagnosis_      527
                           weights
                         WHERE visitid = 1234) AS BTW,
                               trial_condition_tokens AS QT
                        WHERE BTW.token = QT.token
                      GROUP BY BTW.tid, QT.tid) AS SI,          528
              (SELECT *
            FROM      visit_diagnosis_sumweights      529
              WHERE visitid = 1234 ) AS BSUMW,
              (SELECT Q.tid, SUM(BTW.weight) AS sumw      530
            FROM      trial_condition_tokens Q,
                               visit_diagnosis_tokenweights AS BTW
                        WHERE Q.token = BTW.token
                      GROUP BY Q.tid ) AS QSUMW          531
              WHERE BSUMW.tid=SI.tid1 and SI.tid2 = QSUMW.tid ) 532
  SELECT PV.visitid, CT.trialid, s.score          533
  FROM visit AS PV, trial AS CT, scores AS S      534
  WHERE PV.visitid = 1234 AND CT.city='NEW YORK' AND 535
        S.tid1=PV.visitid AND S.tid2=CT.trialid AND 536
        S.score>0.5                                     537

```

### 8.5.2 Semantic Matching Implementation

550

Assume that the synonym and hyponym data are stored in two tables *synonym* and *hyponym* with columns *src* and *tgt*. The column *src* contains *concept IDs* of the terms, and the column *tgt* contains the terms. This is a common approach in storing semantic knowledge, used in NCI thesaurus [24] and Wordnet's synonym sets (*synsets*) [22], for example. Alternatively, these data could be stored in a table *thesaurus* with an additional column *rel* that stores the type of the relationship, or it could even be stored in XML. In the case of XML, *synonym* and *hyponym* can be views defined in a hybrid XML relational DBMS such as DB2. For brevity, we limit our discussion to semantic knowledge stored as relational data, although the framework is easily extensible to other formats. We show the details of the SQL implementation of the *synonym* and *hyponym* native linkage methods in the following two examples.

*Example 8.8 (synonym Native Method).* The following LinQL query considers again patient visit 1234 but now uses synonyms to find matching clinical trials.

563

564

```

SELECT PV.visitid, CT.trialid
FROM visit AS PV, trial AS CT
WHERE PV.visitid = 1234 AND CT.city='NEW YORK' AND
LINK PV.diag WITH CT.cond
USING synonym

```

565  
566  
567  
568  
569

This query is rewritten to:

570

```

SELECT DISTINCT PV.visitid, CT.trialid
FROM trial AS CT, visit AS PV, synonym AS syn
WHERE PV.visitid = 1234 AND CT.city='NEW YORK' AND
(src in (SELECT src
          FROM synonym s
          WHERE s.tgt = CT.cond) )
AND PV.diag = syn.tgt
UNION
SELECT PV.visitid, CT.trialid
FROM trial AS CT, visit AS PV
WHERE PV.visitid = 1234 AND CT.city='NEW YORK' AND
CT.cond = PV.diag

```

571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582

*Example 8.9 (hyponym Native Method).* The following LinQL query considers again patient visit 1234, but now uses hyponyms for the matching:

583  
584

```

SELECT PV.visitid, CT.trialid
FROM visit AS PV, trial AS CT
WHERE PV.visitid = 1234 AND CT.city ='NEW YORK' AND
LINK PV.diag WITH CT.cond
USING hyponym

```

585  
586  
587  
588  
589

This query is rewritten to:

590

```

WITH traversed(src, tgt, depth) AS
  (SELECT src,tgt,1
   FROM hyponym AS ths
   UNION ALL
   (SELECT ch.src, pr.tgt, pr.depth+1
    FROM hyponym AS ch, traversed AS pr
    WHERE pr.src=ch.tgt AND
          pr.depth<2 AND ch.src!='root_node'))
SELECT distinct PV.visitid, CT.trialid
FROM trial AS CT, visit AS PV, hyponym AS ths
WHERE PV.id = 1234 AND CT.city ='NEW YORK' AND
(src in (SELECT distinct src
          FROM traversed tr
          WHERE tr.tgt = CT.cond)) AND
PV.diag = ths.tgt

```

591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605

Note that the hyponym depth is set to 2, but can be customized to any value.

606

### 8.5.3 Effect of Link Index Definitions

607

When a linkindex is defined for a column using a specific linkage method, the 608 above rewriting algorithms will be modified with the goal of speeding up the 609 query execution time at the expense of an extra (offline) preprocessing step. A 610 common indexing strategy for all the native methods is basically materializing 611 parts or all of the view definitions required in their implementation. For exam- 612 ple, in the weightedJaccard implementation of Example 8.7, if a LINKINDEX 613 is defined for visit.diagnosis (using the statement shown in Example 8.6), the 614 following views are materialized: visit\_diagnosis\_tokens, visit\_diagnosis\_weights, 615 visit\_diagnosis\_tokenweights, and visit\_diagnosis\_sumweights. Note that our ini- 616 tial translation algorithms create views in a way that allows effective materialization. 617 For example, the view visit\_diagnosis\_sumweights is defined as an SQL query that 618 simply calculates the sum of the weight of the tokens on-the-fly, only so that this 619 calculation is avoided at run time when the views are materialized. 620

Depending on the DBMS, the views can be materialized manually by creating 621 a table for each view definition, caching the results as a preprocessing step and 622 keeping them fresh (possibly using triggers), or automatically using DBMS support 623 for materialized views. For example, the views can be defined as materialized query 624 tables (MQT) in DB2. Using the optional FULL argument, appropriate database 625 indices are also defined over the target columns and their related (manually) 626 materialized views (tables). Other more specific linkindex definitions are also 627 possible depending on the linkage method, such as specific hash tables as discussed 628 in Sect. 8.4.4. Refer to LinQuer's documentation page for a detailed description of 629 the possible index definitions and their parameters for each native method [30]. 630

### 8.5.4 Other Enhancements

631

So far we have only discussed the core algorithms for translation of a simple LinQL 632 query with no additional arguments. As discussed in previous section, LinQL allows 633 definition of several *optional* arguments. Most of these arguments are link method- 634 specific, and used for setting parameters such as the value of  $q$  for q-gram-based 635 methods, the maximum length of the string, the similarity threshold, and the name 636 of the ontology table and its columns. Another link method-specific argument is the 637 name of a (manually created) weight adjustment table as described in Sect. 8.4.2. 638

Another type of optional argument is the linklimit argument. Although each 639 native linkage method results in a distinct query which can be evaluated indepen- 640 dently, when the user sets a global linklimit value, the queries of different native 641 methods can *cooperate* to optimize performance by avoiding running potentially 642

expensive linkage queries when there are already enough links returned to the user. 643  
 To achieve such a cooperation, the LINKCLAUSEEXPR2SQL algorithm needs to be 644  
 modified. A key assumption we make here is that the sequence of the specifications 645  
 reflects their importance, i.e., the specification that appears first must be evaluated 646  
 first. Therefore, a condition will be added at the end of each SQL code to limit 647  
 the number of results returned (using, e.g., MySQL's LIMIT). Then, in Algorithm 648  
 LINKCLAUSEEXPR2SQL, in addition to using UNION, another condition should be 649  
 added at the end of each query (in the WHERE clause) to check for the number of 650  
 links returned by the previous query. Note that this may result in performance gain 651  
 if one subquery is not evaluated more than once, which is the case in most DBMSs. 652

## 8.6 Case Study

653

In this section, we illustrate the flexibility of the LinQuer framework by applying 654  
 it in a variety of linkage scenarios. We use these scenarios to justify the choices 655  
 we have made in developing LinQuer. These scenarios are built around an online 656  
 database of clinical trials published on ClinicalTrials.gov [28]. This database is a 657  
 registry of federally and privately supported clinical trials conducted in research 658  
 centers all around the world. It contains detailed information about the trials, 659  
 including information about the medical conditions associated with the trials, their 660  
 eligibility criteria, and locations. 661

### 8.6.1 Datasets

662

The clinical trials' database used in our experiments contains 111,227 trials. We 663  
 retrieved the data in XML format, and transformed the data into relational (and 664  
 RDF) using the xCurator framework [27], as a part of the LinkedCT project [18]. 665  
 Other datasets that we used in our experiments include a database of patient visits 666  
 or Electronic Medical Records (EMR), and a database of DBpedia [6] objects of 667  
 type *disease* and *drug*. We also used the National Cancer Institute's (NCI) thesaurus 668  
 [24] as a source of semantic information about medical terms. Detailed statistics on 669  
 these datasets is shown in Table 8.1.<sup>1</sup> 670

Due to privacy issues associated with EMR data, our patient visits database is 671  
 synthetic, generated using an extension of the UIS database generator [19] that 672  
 picks diagnosis and prescription values from the NCI terms to fill the patient visits 673

---

<sup>1</sup>To make our example queries simple, we assume that the databases are denormalized and we have a single table for clinical trials (*trial*), a table storing patient visits (*visit*), and tables storing DBpedia disease (*dbpedia\_disease*) and drug (*dbpedia\_drug*) data. In reality, the database is normalized and these relations are decomposed into multiple relations.

**Table 8.1** Dataset statistics

Dataset	Entity ( <i>table.attribute</i> )	Count	
Clinical trials	Trial ( <i>trial.trialid</i> )	111,227	t26.1
	Condition ( <i>trial.condition</i> )	25,935	t26.2
	Intervention ( <i>trial.intervention</i> )	149,958	t26.3
	Drug ( <i>trial.drug_intervention</i> )	87,231	t26.4
	Reference ( <i>trial.reference</i> )	86,214	t26.5
Patient visits	Visit ( <i>visit.visitid</i> )	10,000	t26.6
	Diagnosis ( <i>visit.diagnosis</i> )	9,319	t26.7
	Prescription ( <i>visit.prescription</i> )	8,289	t26.8
DBpedia	Disease ( <i>dbpedia_disease.label</i> )	5,154	t26.9
	Drug ( <i>dbpedia_drug.label</i> )	4,658	t26.10
NCI thesaurus	Concept ( <i>ontology.conceptid</i> )	89,129	t26.11
	Term ( <i>ontology.term</i> )	246,511	t26.12
	Synonym Relationship ( <i>synonym.conceptid, synonym.term</i> )	244,192	t26.13
	Hyponym Relationship ( <i>hyponym.conceptid, hyponym.term</i> )	99,553	t26.14
			t26.15
			t26.16
			t26.17

table. Our data generator also randomly injects a small amount of string error in the diagnosis field to resemble real EMR records. The error injected in the string resembles real errors and typos occurring in string databases, for example, replacing a character with an adjacent character on a keyboard, or swapping two characters or word tokens.

### 8.6.2 Effectiveness and Accuracy Results

679

In what follows, we describe several link discovery scenarios involving clinical trials. While the first scenario is described in more detail (including its intermediate steps and corresponding linkage specifications), for the other scenarios we only show the final results and only mention changes to preceding LinQL statements.

*Case 1 (Linking patient visits to trial conditions)* The objective here is to discover links to clinical trials that are related to the conditions of certain patients. For this study, we consider 1,000 random patients from table *visit*, where column *diagnosis* stores the condition associated with a patient's visit. The *trial* table stores the trial condition in its column *condition*. The records linked by a simple exact matching are obtained by the SQL query:

```
SELECT DISTINCT PV.visitid, CT.trialid
FROM visit PV, trial CT
WHERE PV.diagnosis = CT.condition
```

690

691

692

The query links only 46 out of the 1,000 patient visits records, each to an average 693 of 90 trial records, resulting in 4,127 visit-trial links. The reason why only 5% of 694 the patient visits are linked is that very different (string) representations of the same 695 entities (conditions) are used in the two sources, while string errors in *diagnosis* 696 values make the situation worse. As a next step, we try a string matching linkage 697 method starting with a low similarity threshold (expecting this to give a high recall) 698 using the following linkspec and query: 699

```
CREATE LINKSPEC weightedJaccard04
AS weightedJaccard (0.4, 2, 50).
```

700  
701

```
SELECT PV.*, CT.*
FROM visit PV, trial CT
WHERE LINK PV.diagnosis WITH CT.condition
USING weightedJaccard04
```

702  
703  
704  
705

The user can now go through a small subset of the results obtained from the above 706 query to estimate the precision of the returned links with respect to the similarity 707 score. The accuracy of the links is subjective and depends on the application. In our 708 scenario, for example, links from a visit record with diagnosis “Alpa Thalassemia” 709 (misspelled record of “Alpha Thalassemia”) to trials with conditions “Alpha Tha- 710 lassemia,” “ $\alpha$ -Thalassemia,” and “Thalassemia” should be considered correct and 711 we would like to find them. However, linking to a trial with condition “Beta Tha- 712 lassemia” is not a correct link for “Alpha Thalassemia” diagnosis. We obtained the 713 following accuracy results for this scenario by investigating the results of the above 714 query for 100 random distinct *diagnosis* values matched with *condition* values: 715

Threshold	Number of links	Accuracy (Precision)
0.70	22	91%
0.65	36	86%
0.60	63	84%
0.55	104	77%
0.50	182	66%
0.45	303	54%
0.40	579	40%

By choosing a high threshold of 0.70, 22 links are returned out of which 20 716 (91%) are correct. However, by choosing a threshold of 0.4, 579 links are returned 717 (more than five links per diagnosis value in the 100 results), but only 231 (40%) 718 of them are correct. Given these observations, a user can choose the appropriate 719 threshold that works best for the specific linkage needs. For our example, we choose 720

a threshold of 0.55 that returns on average almost one link per diagnosis value, 721  
 and has a reasonable accuracy. Overall, we get 32,919 links for 421 (out of 1,000) 722  
 distinct patient visits. Based on the above evaluation, we can estimate that around 723  
 324 (77%) of the visits will link correctly to a trial's condition. 724

The next step is to use the semantic information in NCI to improve the linkage 725  
 using the LinQL query below: 726

```
SELECT DISTINCT PV.visitid, CT.trialid
FROM visit PV, trial CT
WHERE LINK PV.diagnosis WITH CT.condition
  USING synonym
```

Linking using semantic matching based only on synonyms results in 12,343 links 731  
 from 130 distinct visits. Note that these links are all correct links since they are 732  
 derived using accurate synonyms provided by the NCI thesaurus, and no further 733  
 approximate string matching is performed at this stage. From these, 2,453 links 734  
 from 15 distinct visits could not be found using exact or string matching. 735

Repeating the above query with hyponym linkage method for semantic matching 736  
 based on hyponyms of depth 2 from NCI, results in 15,464 links from 60 visit 737  
 records, out of which 14,571 links from 14 visit records could not be found in 738  
 previous steps. Note that the number of discovered links per visit is much higher for 739  
 the hyponym method when compared to the synonym or string matching methods. 740  
 This not only shows the importance of the hyponym method but also illustrates its 741  
 usefulness in situations like the current setup where trials refer to broad disease 742  
 categories instead of specific disease names (e.g., "Blood Disorder" instead of 743  
 "Beta-Thalassemia"). 744

One reason for the relatively low total number of discovered links by the synonym 745  
 and hyponym methods (linking only 173 out of the 1,000 visit records) is the errors 746  
 present in the *diagnosis* values of the *visit* table. This calls for using a linkspec that 747  
 combines semantic matching (e.g., synonym method) with string matching (e.g., 748  
 weightedJaccard method). The LinQL code to do this is: 749

```
CREATE LINKSPEC mixmatch
AS LINK source WITH target
  USING synonym(synonym,conceptid,term)
  AND
  LINK source WITH synonym.term
  USING weightedJaccard (0.7, 2, 50)

SELECT DISTINCT PV.visitid, CT.trialid
FROM visit PV, trial CT
WHERE LINK PV.diagnosis WITH CT.condition
  USING mixmatch
```

Using a combined string and semantic matching results in 49,931 links from 761  
 363 distinct visits, 28,444 more links from 33 more visit records when compared 762  
 with matching based on synonyms or string matching. In the above linkspec, we 763  
 have chosen a higher similarity threshold for string matching which results in 764  
 highly accurate links based on our manual verification of the results. Depending 765  
 on the results of the above steps, the user can write a single query for the linkage 766  
 needs specific to the application. Here, we choose to combine exact matching, 767  
 string matching, semantic matching based on synonyms and hyponyms, and mixed 768  
 semantic matching allowing string errors. This can all be expressed using the query 769  
 below: 770

```
SELECT DISTINCT PV.visitid, CT.trialid
FROM visit PV, trial CT
WHERE LINK PV.diagnosis WITH CT.condition
```

```
USING weightedJaccard
OR
LINK PV.diagnosis WITH CT.condition
USING synonym
OR
LINK PV.diagnosis WITH CT.condition
USING hyponym
OR
LINK PV.diagnosis WITH CT.condition
USING mixmatch
```

The combined approach results in 73,387 links from 482 visit records to the 784  
 related clinical trials. Overall, we have: 785

Method	Links #	Visits #
1. Exact match	4,127	46
2. String match	32,919	421
3. Synonym match	12,343	130
4. Hyponym match	15,464	60
5. Mixed match	49,931	363
Total (combined)	73,387	482

The above clearly illustrates that the LinQuer framework facilitates the fast 786  
 development and testing of linkage methods. Since LinQuer is fully integrated with 787  
 declarative querying, the accuracy (precision and recall) of linkage methods can 788  
 easily be tested over small portions of the data, or over portions where a user knows 789  
 what links he or she desires. 790

*Case 2 (Linking prescriptions to trial interventions)* Now consider a user who 791  
 wishes to link patients who were prescribed a drug with *all* clinical trials that use 792

that drug. To collect all these trials, the user will again need the results produced from a variety of algorithms. As in the previous case, we start with exact matching, try string matching and tune its threshold, and then use synonym, hyponym and the `mixmatch linkspec` defined above. For string matching using `weightedJaccard` method, we choose threshold 0.6 based on a similar manual inspection of a subset of the results. The table below summarizes the results obtained for linking 1,000 visit records to trials by matching prescription values of the visits to drug intervention values of trial records:

Method	Links #	Visits #
1. Exact match	1,323	93
2. String match	7,200	270
3. Synonym match	16,804	349
4. Hyponym match	145	12
5. Mixed match	20,730	399
Total (combined)	23,528	470

Notice that the `synonym` method is much more effective here (when compared to Case 1), in terms of both the number of records linked and the number of links per record. This is mainly due to the fact that the same drug can be listed under a variety of different names (brand names and scientific names), depending on the institution that performs the trial. Also notice that (like in Case 1) multiple methods do find the same links (the total number of links is less than the sum of the links returned by each method). However, each of the five methods is effective in finding links that cannot be found using any of the other methods. This is highly desirable since the goal is to link as many potential visit records as possible.

*Case 3 (Linking trial conditions to DBpedia diseases) and Case 4 (Linking trial interventions to DBpedia drugs)* In these scenarios, we are seeking links from the clinical trials' condition and interventions fields to the DBpedia disease and drug entities, respectively. Unlike the previous cases, assume here that the user only needs to link to a single DBpedia entity per each condition and drug. This makes sense since in most cases there should be a single record in DBpedia for a single disease (condition) or drug intervention in the trials data. Therefore, the user uses the `LINKLIMIT 1` option in the LinQL query to limit the number of links. Then, when a link based on exact matching is found for a record, there is no need to look for approximate string or semantic matches for that record. This could potentially lead to a significant performance improvement.

The final linkage specification query for linking trial conditions to DBpedia diseases (table `dbpedia_disease`) is as follows. The query for trial interventions to DBpedia drugs is similar.

```

SELECT DISTINCT CT.condition, DBPD.uri
FROM trial CT, dbpedia_disease DBPD
WHERE LINK CT.condition WITH DBPD.label
    USING weightedJaccard
    OR
    LINK CT.condition WITH DBPD.label
    USING synonym
    OR
    LINK CT.condition WITH DBPD.label
    USING hyponym
    OR
    LINK CT.condition WITH DBPD.label
    USING mixmatch
    LINKLIMIT 1

```

Again we choose threshold 0.6 for weightedJaccard method based on investigation of the accuracy of the links for a random subset of the conditions. The table below summarizes the results for different steps of the linkage from 1,000 condition and drug interventions:

Method	Disease links#	Drugs links#
2. Exact match	22	239
3. String match	239	393
4. Synonym match	26	333
5. Hyponym match	46	0
5. Mixed match	101	347
Total	292	422

Notice the obvious need for allowing mixed string and semantic matching in these two cases. The trials source, NCI thesaurus and DBpedia/Wikipedia names all use different conventions and therefore there are cases where strings do not exactly match. For example, “Adenocarcinoma of Esophagus” in trials matches with “Carcinoma of Esophagus,” synonym of “Esophageal Cancer” in the thesaurus which matches with “Esophageal\_cancer” in DBpedia. Notice that though we are combining many methods, the LinQL LINKLIMIT 1 option allows us to (efficiently) return the best match for each record.

*Case 5 (Finding related trials)* To show the flexibility of the LinQuer framework, we investigate its effectiveness in a rather different scenario. In this case, the goal is linking trials that are related to each other. Different attributes and measures can be used to identify trials that are related. In this experiment, we use the *reference* attribute of the trials and consider two trials related if the title and authors of their associated references (publications) are similar. Our trials database stores references in a single long text record that includes the names of the authors, title of the paper,

the conference or journal and the date of the publication. Therefore, in order to find 857 similarity we do not have any relevant semantic information to exploit. Furthermore, 858 we are not interested in typos and different representations of the same string 859 here. Instead, the similarity function should measure the amount of co-occurrence 860 of (important) words in the two strings. The following weightedJaccard linkspec 861 performs linkage based on word tokens: 862

```
CREATE LINKSPEC wordTokenJaccard
AS weightedJaccard (0.7, 0, 100)
```

863

864

Using the linkspec over 10,000 random trials results in 667 links between these 865 trials, whereas exact matching results in only seven links. Note that here we assume 866 that similarity score above 0.7 between the reference strings indicates correlation 867 between the trials, and we have verified that this assumption is reasonably accurate 868 by manually going through a subset of the links. 869

### 8.6.3 Effectiveness of Weight Tables

870

In what follows, we briefly show the effectiveness of the functionality enhancement 871 we proposed based on manual definition of a weight-adjustment table by the user. 872 Assume that the user defines the following simple weight table: 873

String	Weight
“Syndrome”	0.4
“The”	0.1
“Disorder”	0.2
“Disease”	0.2

We repeat the experiment for Case 1 (linking to trial conditions from a database 874 of patient visits) with updated weight tables based on the above input weight 875 adjustment table. This results in lower weights for the tokens of the above strings 876 in the weight tables (views) of *visit.diagnosis* and *trial.condition* columns. String 877 matching with the same settings, i.e., using weightedJaccard similarity function 878 with threshold 0.55 on 1,000 random base records, results in 121 additional links 879 out of which 91 are correct (accuracy 75%) and drops 63 of the links found with no 880 weight adjustments out of which 15 were wrong links. This means that overall, the 881 linkage has resulted in 58 more links with roughly the same accuracy as the case 882 with no adjustments. 883

Note that we obtained these results by choosing the weight adjustment values 884 in the above simple table based only on our domain knowledge, and we have 885 not varied the values to obtain the best results. What is more important is that 886

we can use this extensible technique and leverage weight values to improve the 887 accuracy of the link discovery, as a result of the SQL-based implementation method. 888 The implementation of the weighted Jaccard similarity function and the above 889 customization of weights using a UDF, rather than a native method, could be quite 890 complex and inefficient. 891

### 8.6.4 Performance Results

892

Here, we report running times of the above examples to show the performance of 893 the system. Note that since our focus in this chapter has been on the functionality 894 of the framework, we have not applied specific hashing techniques (some of which 895 we discussed briefly in Sect. 8.4) to improve efficiency. We also do not compare the 896 benefits of possible indexing and hashing strategies. We ran the experiments on a 897 Dell PowerEdge R310 Server with 24 GB memory and Intel X3470 Xeon Processor 898 2.93 GHz (8MB Cache, Turbo HT), running Ubuntu 10.04.3 LTS (64-bit), and IBM 899 DB2 9.7 Data Server Trial Edition. To obtain statistical significance, we report the 900 average time from several runs of each experiment. 901

We first report the running times for `LINKINDEX` statements over the attributes 902 and link methods used in all the cases described in this section. For Case 1 (linking 903 visits to trial conditions), for example, we define `LINKINDEX` over `visit.diagnosis`, 904 `trial.condition` and `synonym.term` columns using `weightedJaccard` method as 905 follows: 906

```
CREATE LINKINDEX FULL FOR visit.diagnosis USING weightedJaccard; 907
CREATE LINKINDEX FULL FOR trial.condition USING weightedJaccard; 908
CREATE LINKINDEX FULL FOR synonym.term USING weightedJaccard; 909
```

As stated earlier, the above statements translate into a set of preprocessing queries 910 that materialize the views for tokenization and token weight calculation required 911 for the final linkage queries. With the `FULL` argument, database indices are also 912 built over the result tables. It is also possible to use the `HALF` option that will result 913 only in materialization of the tokenization views. This is useful for more dynamic 914 tables, and when the attribute is used only as target (and not source) in the LinQL 915 queries, where token weights are not needed. Figure 8.3 shows the running time 916 for tokenization queries (that materialize token views), weight table queries (that 917 materialize token weight views), and the time for building database indices for token 918 and weight tables, for all the attributes and link methods used in our study. 919

Notice that the overall preprocessing times are less than a minute for small tables, 920 and a few minutes for larger ones. Word tokenization for references (case 5) takes 921 the longest time due to the more expensive tokenization query and longer average 922 string size (the average *reference* string length is 226, whereas as the average length 923 of ontology terms is 25 characters). 924

	Tokenization Queries	Tokenization Indexing	Weight Table Queries	Weight Table Indexing	Total
<i>trial.condition</i>	3.4	66.1	9.0	69.5	148.0
<i>trial.drug_intervention</i>	9.7	146.1	113.1	119.0	387.9
<i>visit.diagnosis</i>	1.2	21.1	3.6	21.8	47.7
<i>visit.prescription</i>	0.7	12.8	4.5	12.2	30.2
<i>dbpedia_drug.label</i>	0.4	8.4	3.0	7.8	19.6
<i>dbpedia_disease.label</i>	3.6	21.5	6.2	15.6	46.9
<i>trial.reference</i>	58.7	547.7	43.6	696.7	1346.7
<i>synonym.term</i>	84.6	171.3	64.6	227.0	547.5

**Fig. 8.3** Preprocessing (indexing) time (in seconds)

Other linkage methods can also take advantage of `LINKINDEX` definitions. For 925 the `hyponym` method and a given depth, for example, the view for traversing the 926 ontology tree can be materialized in advance. For our ontology table and for our 927 cases (using depth 2), this can be done using the statement below, which takes just 928 below 53 min to run. For the custom link method `mixmatch` over *visit.diagnosis*, 929 for example, defining `LINKINDEX FULL` as shown below will result in materializing 930 the scores view between the ontology terms and diagnosis values using the 931 `weightedJaccard` method with the given parameters, which takes around 42 min 932 in this case. Note that these are expensive preprocessing queries but can speed up 933 running time significantly when the query needs to be repeated many times, and 934 when the indexed relations do not change frequently. In our scenario, NCI thesaurus 935 is updated monthly, and the clinical trials are updated once every night, which 936 will allow enough time to repeat the preprocessing. Of course, better index update 937 strategies can be used, and depending on the DBMS, the view materialization can 938 be automated, using Materialized query tables (MQTs) in DB2, for example. 939

```
CREATE LINKINDEX USING hyponym(hyponym,term,2); 940
CREATE LINKINDEX FULL FOR visit.diagnosis USING mixmatch; 941
```

The table below shows the running time (in seconds) for all the cases and 942 link methods described in this section. Note that these are running times for 943 batch processing of 1,000 base records in each case. The average running time of 944 each method for linking a single record is under a second (a few milliseconds in 945 most cases), which shows the suitability of the methods for online link discovery 946 scenarios. Also note that due to the above `LINKINDEX` statements, `hyponym` and 947 `mixmatch` methods take only slightly longer than the `synonym` method. The string 948 matching, however, is the most expensive step. This calls for more advanced hashing 949 and indexing strategies for larger databases (see Sect. 8.4 for a brief discussion). 950

	Case 1	Case 2	Case 3	Case 4	Case 5
Exact match	0.01	0.02	<0.01	<0.01	0.27
String match	187.10	240.20	38.32	16.97	672.23
Synonym match	0.12	0.22	0.13	0.11	N/A
Hyponym match	0.21	0.22	0.19	0.19	N/A
Mixed match	0.21	0.31	0.11	0.11	N/A

## 8.7 Related Work

In terms of the overall framework, the work described in this chapter is closely related to the work on declarative data quality and cleaning [5, 13, 16]. When compared with all the existing techniques in this area, a distinctive feature of the LinQuer framework is its focus on enabling the discovery of any type of semantic link, not just the *same-as* link that is the result of entity resolution or duplicate detection. Another closely related framework is the work of Das et al. [10], in which a set of SQL operators are proposed to support ontology-based semantic linking over relational data. A key advantage of the LinQuer framework is allowing string matching along with semantic matching which is crucial in many real-world scenarios. Moreover, the LinQL specification language allows the definition of new operators, which could be a mix of several semantic and string matching operators. Overall, LinQuer complements and extends the above-mentioned frameworks by providing a framework for fast prototyping and testing of semantic and syntactic matching for link discovery. LinQuer can clearly be used in combination with a nonrelational framework for link discovery such as SILK [26]. Whereas LinQuer discovers links over relational data *before* these are published as RDF, SILK focuses on the discovery of semantic links *after* the RDF publication occurs. Another key difference between the two systems is that while LinQuer is an extensible framework that employs several semantic and syntactic methods particularly suitable for string data, SILK offers a limited set of attribute-similarity measures that can be combined in various ways for link discovery.

As stated earlier, one of the main applications of the LinQuer framework is to enhance Web data publishing. There are many methodologies for generating RDF views over relational data. These methodologies are often based on declarative specification of the mapping between relational tables and RDF triples. These frameworks include, but are not limited to, D2RQ and D2R Server [7], Openlink Virtuoso [12], and Triplify [3]. The success of these tools motivates a similarly declarative framework for link discovery such as LinQuer so that the published data sources can be enriched internally and linked to other data sources to enhance data sharing.

## 8.8 Conclusion

982

In this chapter, we presented a declarative extensible framework for link discovery 983 from relational data. We presented LinQL, an extension of SQL for specification 984 of linkage requirements for linking relational records, along with the details of 985 its implementation. We showed how this framework adopts and extends existing 986 syntactic and semantic matching techniques, and described a few functionality 987 enhancements specifically designed for this framework. We showed the effective- 988 ness of this approach in several link discovery scenarios in a real-world health 989 care application. Our focus in this chapter has been on efficient techniques that can 990 handle large datasets, but also on usability. We showed how a user can interactively 991 experiment with and customize different link methods to better understand what 992 are the most effective methods for her domain. We believe that this framework 993 can significantly simplify and enhance the process of publishing a high-quality data 994 source with links to other data sources on the Web. A user/data publisher can use 995 this framework to easily find the appropriate linkage algorithms for the specific 996 domain, as well as the optimal value of the required parameters. Combined with an 997 existing popular declarative approach for generating linked data on the Web such 998 as D2R Server [7], this can lead to a quick and simple way of publishing an online 999 data source with high-quality links. This will in turn enable or enhance effective 1000 semantic search over the data source and the Web of Data. 1001

**Acknowledgements** This work has been partially supported by the NSERC Business Intelligence 1002 Network. Hassanzadeh has been supported by an IBM Graduate Fellowship. We thank Reynold S. 1003 Xin for implementation of the LinQuer API and Web interface, improving the overall design of the 1004 system and the LinQL grammar. 1005

## References

1006

- Appelt, D.E.: Introduction to information extraction. *AI Commun.* **12**(3), 161–172 (1999) 1007
- Arasu, A., Ganti, V., Kaushik, R.: Efficient exact set-similarity joins. *Proceedings of the 1008 International Conference on very Large Data Bases (VLDB)*, pp. 918–929 (2006) 1009
- Auer, S., Dietzold, S., Lehmann, J., Hellmann, S., Aumueller, D.: Triplify: light-weight 1010 linked data publication from relational databases. *International World Wide Web Conference 1011 (WWW)*, pp. 621–630 (2009) 1012
- Bayardo, R.J., Ma, Y., Srikant, R.: Scaling up all pairs similarity search. *International World 1013 Wide Web Conference (WWW)*, pp. 131–140. Banff, Canada (2007) 1014
- Bilke, A., Bleiholder, J., Böhm, C., Draba, K., Naumann, F., Weis, M.: Automatic data fusion 1015 with HumMer. *Proceedings of the International Conference on very Large Data Bases (VLDB)*, 1016 pp. 1251–1254 (2005) 1017
- Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: 1018 DBpedia – a crystallization point for the web of data. *J. Web Semant.* **7**(3), 154–165 (2009) 1019
- Bizer, C., Seaborne, A.: D2RQ – treating non-RDF databases as virtual RDF graphs. 1020 *Proceedings of the International Semantic Web Conference (ISWC)* (2004) 1021
- Cohen, W.W.: Data integration using similarity joins and a word-based information represen- 1022 tation language. *ACM Trans. Inf. Syst.* **18**(3), 288–321 (2000) 1023

9. Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A comparison of string distance metrics for name-matching tasks. Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), pp. 73–78. Acapulco, Mexico (2003) 1024  
1025  
1026
10. Das, S., Chong, E.I., Eadon, G., Srinivasan, J.: Supporting ontology-based semantic matching in RDBMS. Proceedings of the International Conference on very Large Data Bases (VLDB), pp. 1054–1065 (2004) 1027  
1028  
1029
11. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2007) 1030  
1031
12. Erling, O., Mikhailov, I.: Virtuoso: RDF support in a native RDBMS. Semantic Web Information Management, pp. 501–519. Springer, Berlin, Heidelberg, New York (2009) 1032  
1033
13. Galhardas, H., Florescu, D., Shasha, D., Simon, E., Saita, C.A.: Declarative data cleaning: language, model, and algorithms. Proceedings of the International Conference on very Large Data Bases (VLDB), pp. 371–380 (2001) 1034  
1035  
1036
14. Galperin, M.Y., Cochrane, G.: The 2011 nucleic acids research database issue and the online molecular biology database collection. Nucleic Acids Res. **39**(Database-Issue), 1–6 (2011) 1037  
1038
15. Gravano, L., Ipeirotis, P.G., Jagadish, H.V., Koudas, N., Muthukrishnan, S., Srivastava, D.: Approximate string joins in a database (almost) for free. Proceedings of the International Conference on very Large Data Bases (VLDB), pp. 491–500 (2001) 1039  
1040  
1041
16. Hassanzadeh, O.: Benchmarking declarative approximate selection predicates. Master's Thesis, University of Toronto, Toronto, Ontario, Canada (2007) 1042  
1043
17. Hassanzadeh, O., Kementsietsidis, A., Lim, L., Miller, R.J., Wang, M.: A framework for semantic link discovery over relational data. Proceedings of the Conference on Information and Knowledge Management (CIKM), pp. 1027–1036 (2009). URL <http://dx.doi.org/10.1145/1645953.1646084> 1044  
1045  
1046  
1047
18. Hassanzadeh, O., Kementsietsidis, A., Lim, L., Miller, R.J., Wang, M.: LinkedCT: a linked data space for clinical trials. CoRR **abs/0908.0567** (2009) 1048  
1049
19. Hernández, M.A., Stolfo, S.J.: The merge/purge problem for large databases. ACM SIGMOD international conference on the management of data, pp. 127–138 (1995) 1050  
1051
20. Indyk, P., Motwani, R., Raghavan, P., Vempala, S.: Locality-preserving hashing in multidimensional spaces. ACM Symposium on Theory of Computing (STOC), pp. 618–625 (1997) 1052  
1053
21. Kementsietsidis, A., Lim, L., Wang, M.: Supporting ontology-based keyword search over medical databases. Proceedings of the AMIA 2008 Symposium, pp. 409–13. American Medical Informatics Association (2008) 1054  
1055  
1056
22. Miller, G.A.: WordNet: a lexical database for English. Commun. ACM **38**(11), 39–41 (1995) 1057
23. Naumann, F., Herschel, M.: An introduction to duplicate detection. Synthesis Lectures on Data Management. Morgan and Claypool Publishers, Seattle, WA USA (2010) 1058  
1059
24. Sioutos, N., de Coronado, S., Haber, M.W., Hartel, F.W., Shaiu, W., Wright, L.W.: NCI thesaurus: a semantic model integrating cancer-related clinical and molecular information. J. Biomed. Inform. **40**(1), 30–43 (2007) 1060  
1061  
1062
25. Suchanek, F.M., Kasneci, G., Weikum, G.: YAGO: a large ontology from Wikipedia and WordNet. J. Web Semant. **6**(3), 203–217 (2008) 1063  
1064
26. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and maintaining links on the web of data. Proceedings of the International Semantic Web Conference (ISWC), pp. 650–665 (2009) 1065  
1066
27. Yeganeh, S.H., Hassanzadeh, O., Miller, R.J.: Linking semistructured data on the web. Proceedings of the International Workshop on the Web and Databases (WebDB) (2011) 1067  
1068
28. ClinicalTrials.gov, A Service of the US National Institutes of Health – <http://clinicaltrials.gov/> (2011). Accessed 28 July 2011 1069  
1070
29. State of the LOD Cloud. Version 0.2. <http://www4.wiwiss.fu-berlin.de/lodcloud/state/> (2011). Accessed 28 July 2011 1071  
1072
30. The LinQuer Project - <http://purl.org/linquer> 1073

**AUTHOR QUERIES**

- AQ1. Kindly check the corresponding author.
- AQ2. Please confirm the insertion of apostrophe for ‘clinical trials database’ update in the other occurrence also.
- AQ3. please confirm the insertion of ‘he or’ in the sentence “For example, she may...”.
- AQ4. Please confirm the insertion of ‘or her’ in the sentence “Suppose a user writes a UDF...”.
- AQ5. In the sentence “...can easily be tested over...” ‘she’ has been changed as ‘he or she.’ Please check.
- AQ6. Kindly provide the accessed date for Ref. [30].

UNCORRECTED PROOF

# Chapter 9

## Embracing Uncertainty in Entity Linking

Ekaterini Ioannou, Wolfgang Nejdl, Claudia Niederée, and Yannis Velegrakis

### 9.1 Introduction

The modern Web has grown from a publishing place of well-structured data and HTML pages for companies and experienced users into a vivid publishing and data exchange community in which everyone can participate, both as a data consumer and as a data producer. Unavoidably, the data available on the Web became highly heterogeneous, ranging from highly structured and semistructured to highly unstructured user-generated content, reflecting different perspectives and structuring principles. The full potential of such data can only be realized by combining information from multiple sources. For instance, the knowledge that is typically embedded in monolithic applications can be outsourced and, thus, used also in other applications [10]. Numerous systems nowadays are already actively utilizing existing content from various sources such as WordNet or Wikipedia. Some well-known examples of such systems include DBpedia, Freebase, Spock, and DBLife.

A major challenge during combining and querying information from multiple heterogeneous sources is *entity linkage*, i.e., the ability to detect whether two pieces of information correspond to the same real-world object [19, 28]. The task is also important in data cleaning applications [13] and can be found in the literature under different names, such as merge-purge [23], entity identification [29], deduplication [33], data matching [8, 15], reference reconciliation [17], or resolution [5]. This topic

---

AQ1

E. Ioannou (✉)

Technical University of Crete, University Campus – Kounoupidiana, 73100 Chania, Greece  
e-mail: [ioannou@softnet.tuc.gr](mailto:ioannou@softnet.tuc.gr)

W. Nejdl · C. Niederée

L3S Research Center, Appelstr. 9a, 30167 Hannover, Germany  
e-mail: [nejdl@L3S.de](mailto:nejdl@L3S.de); [niederee@L3S.de](mailto:niederee@L3S.de)

Y. Velegrakis

University of Trento, Via Sommarive 14, 38123 Trento, Italy  
e-mail: [vegias@disi.unitn.eu](mailto:vegias@disi.unitn.eu)

has received considerable research attention with many interesting results relying on different methodologies, such as string similarity metrics [8, 9], entity inner-relationships [17, 27], and clustering [7].

Unfortunately, existing approaches for entity linkage assume that data are relatively static. Thus, they typically perform data processing off-line in order to have the results readily available at query time. To achieve this, existing approaches first collect matching evidence, such as similarities between the entity strings or inner-relationships between entities, and based on them, generate information to link the entities, then use predefined thresholds or human intervention to merge the entities. Queries are processed over the resulted merged entities. In modern Web applications, where data may at any time change not only their syntax or structure but also their semantics [35], these techniques are so effective or efficient [19]. This calls for entity linkage techniques that consider and deal with the special characteristics of such data.

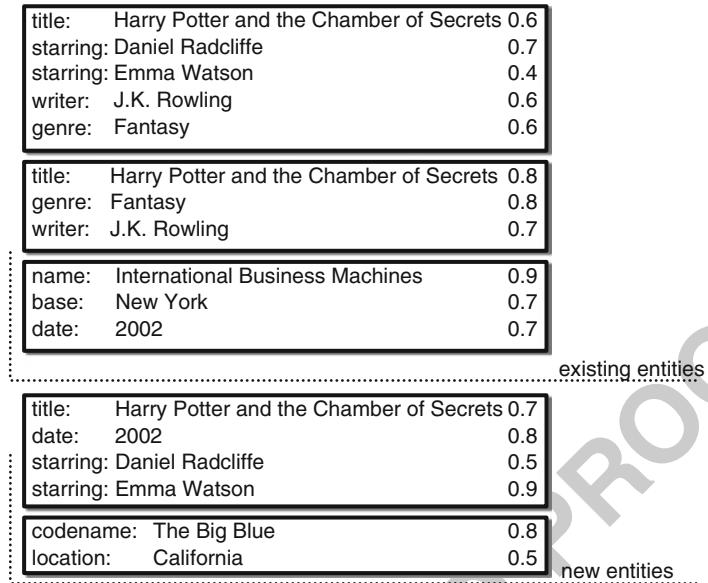
This chapter introduces a novel approach for addressing the entity linkage problem for heterogeneous, uncertain, and volatile data. In the following paragraphs, we present the motivation for this work (Sect. 9.1.1), followed by a discussion of the related challenges (Sect. 9.1.2). We then provide an overview of the approach introduced in this chapter (Sect. 9.4), summarize its contributions (Sect. 9.1.4), and finally present the structure of the chapter (Sect. 9.1.5).

### 9.1.1 Motivation

Consider a system created for monitoring and integrating data from multiple heterogeneous data sources on the Web. The basic data exchange unit of the system is an entity, composed of an identifier and a number of attribute name–value pairs describing the properties of the real-world object the entity represents.

The first part of Fig. 9.1 illustrates three entities existing in the system. The top two entities are referring to the story of Harry Potter and the Chamber of Secrets. The first entity has been extracted through text analysis of Wikipedia articles. Since entity extraction from text is not always accurate, the extracted entity attributes are accompanied with some probabilities reflecting the amount of confidence on the existence of these attributes. In the figure, this confidence is illustrated by the numbers next to the attribute values. The second entity has been extracted from a set of online bookstore databases. A number of these databases contain outdated or inconsistent data; thus, the attributes of the entity are also probabilistic. Finally, the third entity has been extracted by a corpus of news archives. For reasons similar to those of the first entity, its attributes have also some confidence associated with them.

Since the system needs to handle volatile data, we expect continuous appearance of new entities and these entities that need to be integrated with the data already present in the system. The second part of Fig. 9.1 illustrates two additional entities, which the system needs to integrate. Similar to the entities already existing in the



**Fig. 9.1** A small fraction of new entities that should be integrated with the entities already existing in the system. Entities are modeled as a set of attributes, i.e., name–value pairs, each with some confidence value that indicates our belief that this attribute describes the specific entity

system, these two entities are also modeled as a set of attribute name–value pairs, 64  
each with some confidence value. 65

A traditional entity linkage methodology [19] would simply use a predefined 66  
threshold and accept the merging of these entities when their computed similarity is 67  
above this threshold. For the entities of Fig. 9.1, this might mean merging the first 68  
two entities. In this situation, a new entity would be created using the data from 69  
both entities, which might also involve the removal of some name–value pairs when 70  
these are considered as redundant, conflicting, or replicas. 71

There are two main issues with the traditional entity linkage methodologies. The 72  
first is that the system will return no results if asked to return an entity described 73  
using some of the attributes that were removed during the merging. The second 74  
is that the arrival of a new entity might cause the system to get into a stage that 75  
does not accurately reflect reality, since the system is now limited to two options: 76  
either decide that the new entity describes the same real-world object as one of 77  
those that already exist in the system or that the new entity is not among the already 78  
existing entities. This unfortunately ignores the possible options that would arise 79  
from reviewing any of the previous merging decisions. As an example, consider a 80  
system that has previously performed a merging between entities  $e_a$  and  $e_b$ , and 81  
that it now needs to process the new entity  $e_c$ . Merging entity  $e_c$  with  $e_a$  might 82  
provide a better solution than merging it with the previously merging of  $e_a$  with  $e_b$ . 83  
An alternative methodology for considering all possible options would of course be 84

to maintain the original entities and, at each addition, reexecute an entity linkage 85  
technique over the original entities, ignoring the results from previous executions. 86  
Unfortunately, this approach has a prohibitively high computational cost, which 87  
means that it cannot be applied. 88

### 9.1.2 Challenges

89

To create an effective and efficient solution for the entity linkage problem, we need 90  
to consider the characteristics as well as the resulting challenges that appear in the 91  
scenarios we are focusing on (Sect. 9.1.1). This can be summarized as follows: 92

*Challenge 1—Volatile Data.* Collections created by combining data from various 93  
Web applications or extracted data describing resources may constantly change 94  
and evolve through interactions with users or external applications. Therefore, the 95  
knowledge available to the entity linkage techniques is subject to data reduction, 96  
addition, and modification. This implies the need for supporting an incremental 97  
computation and adaptation of the linkage information. 98

*Challenge 2—Heterogeneous Information.* Effectively addressing the entity link- 99  
age problem implies the ability to handle highly heterogeneous data. Dealing with 100  
heterogeneity is a task that touches a number of aspects. For instance, the data model 101  
for the entities should be able to also capture the possible data heterogeneity. In 102  
addition, we need to consider that we are using an entity linkage technique that 103  
handles heterogeneous information. The most common methodology to detect entity 104  
linkages is based on observing similarities between the attribute values from the 105  
entities. However, this assumes that entities describing the same real-world objects 106  
would have the same, or at least similar, attribute values. Another methodology 107  
relies on identifying and facilitating semantic information, such as relationships 108  
between the entities. For example, coauthoring relationship in publications increases 109  
the belief that two authors describe the same object. Our solution should therefore 110  
allow the combination of results generated by various entity linkage techniques, as 111  
a way to capture different linkage methodologies. 112

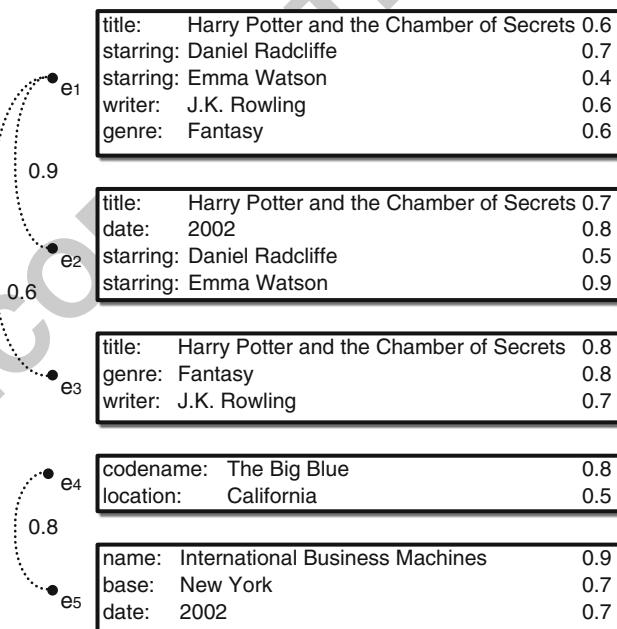
*Challenge 3—Data Uncertainty.* Apart from the uncertainty in the linkage informa- 113  
tion, data uncertainty also appears for other reasons. One example is data uncertainty 114  
that comes directly from the extraction process due to the very low quality that 115  
typically accompanies the unstructured data of such applications [21]. Another 116  
example is the uncertainty introduced when building structures for processing the 117  
data, e.g., social network analysis [1]. These approaches typically affect the quality 118  
of data, which is then reflected through probabilities. Unfortunately, incorporating 119  
uncertainty in a system may break a number of assumptions that many entity linkage 120  
techniques rely upon. Thus, performing entity linkage over uncertain data is a major 121  
challenge. 122

### 9.1.3 Summary of the Approach

123

The methodology we follow takes into consideration heterogeneity, uncertainty, and the volatile nature of the data. It is based on maintaining the linkage information among the entities. As an example, let us consider the entities in Fig. 9.1. It is easy to see that the first two entities may represent the same real-world object, for instance, the first entity may represent the actual movie, whereas the second entity is a DVD with the respective movie. Given that we do not have enough evidence to support a definite decision on whether these entities represent the same real-world object or not, we do not perform a merging between them. We compute and store a probabilistic linkage connecting the two entities. The addition of the new entities requires only the computation of the linkages or (in some cases) the recomputation of the probability of existing linkages.

Figure 9.2 illustrates the computed linkages through the interconnecting dotted lines and alongside their probabilities. As depicted in the figure, these entities have three probabilistic linkages, two among the movie entities that are labeled  $e_1-e_3$ , and one among the company entities that are labeled as  $e_4$  and  $e_5$ . Once we reach a final decision that two or more entities are linked, we can replace them by an equivalent entity consisting of the union of their attributes.

135  
136  
137  
138  
139  
140

**Fig. 9.2** Entities and their probabilistic linkage information, which is shown with the *dotted lines*

Consider now a user looking for the IBM consulting corporation. As is typically 141  
 the case in dataspaces [22], queries are expressed as a series of attribute name–value 142  
 pairs. Thus, the user sends the following query to the system: 143

$\langle \text{name} = \text{"International Business Machines"}, \text{base} = \text{"New York"} \rangle$  144

Clearly among the five entities  $e_1, e_2, e_3, e_4$ , and  $e_5$ , only the fourth satisfies these 145  
 two conditions. Of course, since the attributes of the specific entity exist with some 146  
 uncertainty, specified by the respective probabilities, the existence of the entity in 147  
 the query answers should also be probabilistic. A significant amount of research has 148  
 been carried out in the area of the probabilistic databases [12] on specifying the 149  
 semantics and on the development of efficient query answering techniques for this 150  
 kind of scenarios. 151

Assume now that a user is interested in the works of J.K. Rowling in the year 152  
 AQ3 2002. He or she sends to the system the following query: 153

$\langle \text{writer} = \text{"J.K. Rowling"}, \text{year} = \text{"2002"} \rangle$  154

None of the three entities in Fig. 9.2 contain both attribute name–value pairs as 155  
 specified in the query; thus, any probabilistic database approach will return an 156  
 empty set as an answer. However, the linkage information between entity  $e_1$  and 157  
 $e_2$  indicates that they may represent the same real-world object. If they do, then they 158  
 can be both merged into one entity, say  $e_{12}$  that contains as attributes the union of 159  
 the attributes  $e_1$  and  $e_2$ . That entity will satisfy both the conditions of the last query 160  
 and should be part of the answer set, even though it is not one of the three entities 161  
 that are actually stored in the repository. 162

In a similar situation, assume that the user sends the query: 163

$\langle \text{writer} = \text{"J.K. Rowling"}, \text{genre} = \text{"Fantasy"} \rangle$  164

Answer to the query should take into consideration all the different cases that may 165  
 exist based on the entity linkages. In particular, a complete answer should contain 166  
 three entities, namely, entity  $e_1$ , entity  $e_3$ , and the entity  $e_{13}$  which is the merging of 167  
 entities  $e_1$  and  $e_3$ . Each of these entities should of course be in the answer set of the 168  
 query with some degree of belief, based on the belief of the linkages and the belief 169  
 of the attributes *writer* and *genre*. 170

The answer set for this query could also contain entities  $e_{12}$  and  $e_{123}$ , which are 171  
 created by the merging of entity  $e_2$  with  $e_1$ , and entity  $e_2$  with  $e_1$  and  $e_3$ , respectively. 172  
 Included in the merging, the attributes of  $e_2$  will create entities that have additional 173  
 attributes, such as  $\text{date} = 2002$ . We consider such additional attributes as redundant, 174  
 since the user did not request them through the query. Our basic principle is that 175  
 we do not want to produce results that are not required, and therefore, no merging 176  
 should take place unless it is justified by the query given by the user, the linkages, 177  
 and the attributes composing the entities. 178

Our approach creates the entity mergings by using available entity linkages. 179  
 Since the linkages are probabilistic, for an effective query mechanism, we need 180  
 to take into consideration all the different combinations that may occur. Each such 181  
 combination will partially contribute to the answer set. However, materialization of 182

all combination will lead to exponential increase of the data, which is inefficient to 183  
generate and store. Instead, query processing at runtime takes into consideration the 184  
related probabilistic linkages; computes the different combinations, along with their 185  
respective probability of existence; and then generates the answer set by merging 186  
the data produced from each combination. 187

### 9.1.4 Contributions

188

The main focus of this chapter is to efficiently and effectively address the entity 189  
linkage problem as this appears in heterogeneous, uncertain, and volatile data. This 190  
is achieved by allowing data integration systems to maintain probabilistic linkage 191  
information and perform entity-aware query processing over the data and thus 192  
retrieve answers to queries that reflect the corresponding real-world objects. This 193  
methodology avoids pitfalls that may result from the one-time a priori merging 194  
decisions, as performed by traditional entity linkage techniques. Furthermore, it can 195  
support highly volatile data more efficiently. The reason is that since no merging 196  
decisions have taken place, the only updates required are on the linkages related to 197  
new data incorporated in the system, or modified data. 198

In an effort to address the entity linkage problem for volatile data, we introduce 199  
a model for representing entities and linkages that aims at bringing together two 200  
worlds: the world of entity linkage and the world of probabilistic databases. The 201  
novelty of this data model is that it uses a generic entity-based representation 202  
model for highly heterogeneous data that support the simultaneous representation 203  
of possible linkages between entities alongside the original data, as generated by a 204  
number of the existing entity linkage techniques. This means that no data merging 205  
is performed in advance, but the outcome of the entity linkage algorithms, i.e., the 206  
pairs of entities possibly representing the same real-world object with the belief 207  
of that being true, is stored in the data. The outcome is a database that contains 208  
uncertainty not only on the attributes of the entities but also on their linkages. 209

Relying on the introduced model that contains a set of probabilistic linkages, 210  
we introduce a methodology to efficiently compute the answers for entity queries . 211  
Query answers reflect the entity linkage and entity representation information, with 212  
special emphasis given to the computation of the probabilities of the possible worlds 213  
based on the data and the matching uncertainty. 214

Generating entities by combining probabilistic linkages has several benefits. 215  
First, it produces additional valid query answering results compared to those of 216  
entity linkage and probabilistic databases, which cannot be simulated with previous 217  
techniques. An interesting feature is that reasoning about the entity linkages is done 218  
on the fly, meaning that some query results may not be explicitly represented in the 219  
database but might be a product of the reasoning which is based on the data as well 220  
as on the query conditions, i.e., by considering the union of all the attributes of the 221  
structures to be merged with corresponding probabilities [36]. 222

### 9.1.5 Organization

223

The remaining of this chapter is structured as follows. Section 9.2 presents and 224 discusses existing approaches that are related to the techniques presented in this 225 chapter. Section 9.3 introduces and explains the data model, which includes the 226 representation of entities and linkages, as well as the mechanism for dealing with 227 data uncertainty. Section 9.4 explains the mechanism for dealing with probabilistic 228 linkage information through on-the-fly entity-aware query processing. Section 9.5 229 reports on our experimental evaluation performed on two real-world datasets. 230 Finally, Sect. 9.6 provides conclusions and provides an overview of current and 231 future work. 232

## 9.2 Related Work

233

Most existing techniques for entity linkage focus on the off-line detection and 234 linkage of data referring to the same real-world objects [14, 19, 20]. These techniques 235 deploy a variety of different methodologies and directions. These includes string 236 similarity metrics [8, 9] for computing the matching being the given textual 237 representations of entities, the use of the available inner-relationships between 238 the entities [17, 27], clustering [7], and blocking techniques [30, 31] for reducing 239 the required execution time. However, as already explained in Sect. 9.1, these 240 techniques have major limitations in addressing the entity linkage problem as this 241 appears in current data [19], e.g., data evolution, uncertainty, and incompleteness. 242

Few existing data integration proposals focus on dealing with uncertain linkage 243 information during query processing. More specifically, Dong et al. [18] investigate 244 the use of the probabilistic mappings between the attributes of the contributing 245 sources with a mediated schema. Applying this method on the data from Sect. 9.1.1 246 would have considered the possible mappings between the attribute names as given 247 by contributing sources with a mediated schema  $S$ . This means that “title” attribute 248 of  $e_1$ ,  $e_2$ , and  $e_3$  is mapped to a “Title” attribute from  $S$  with a probability to show 249 the uncertainty of each mapping. Querying the mediated schema  $S$  will be based on 250 these mappings. For example, query “title = Harry Potter...” returns  $e_1$ ,  $e_2$ , and  $e_3$ . 251 However, it does not really reflect the expected answer, since we know that some of 252 the entities are the same, and thus they should be merged accordantly. In fact, the 253 probabilistic schema mappings described in this approach could actually become an 254 input to our approach. 255

The approach presented in [3] is more similar to ours, since their focus is not on 256 the schema information but on the actual data. The authors assume that the duplicate 257 tuples for each entity are given. In our motivating example (Sect. 9.1.1), this means 258 that all tuples which describe the same entity should have the same identifier, e.g.,: 259

Identifier	Entity	Probability
5	$e_1$	$p_1$
5	$e_2$	$p_2$
5	$e_3$	$p_3$

The tuples that represent different entities are considered as independent and the tuples representing the same entity (having the same identifier) as conditionally dependent. The latter means that only one tuple for each identifier can be part of the results. Our proposal does not require this. We explain how entity linkages contain correlations and provide an appropriate solution.

Other related approaches are dataspaces [22] and Trio [2]. The main focus of these approaches is to create database systems that support uncertainty along with inconsistency and lineage. At some extent, these systems also deal with duplicate tuples and uncertain data. Our approach addresses more challenges of heterogeneous data, mainly by considering linkage/matching on the data (not only on schema information), and also correlations between entities.

Another important aspect of our approach is the efficient management of uncertainty in data; a topic that has received a lot of attention recently. Dalvis and Suciu [11] used the notion of possible worlds to introduce query semantics for independent probabilistic data and presented how to efficiently evaluate queries. The approach by Sen et al. [34] moved towards defining and using different correlations, e.g., that existence of one tuple implies or disallows the existence of another tuple.

### 9.3 Data Model

277

To effectively model highly heterogeneous information, we need a simple and flexible model that will be able to represent relational, XML, RDF, and object-oriented data without significant loss of information. We have chosen to go with a graph-based model that is typically used in dataspaces [22]. The main component of the model is an entity which consists of a number of attributes describing its characteristics and a set of associations between the entities. In particular, we assume the existence of an infinite set of entity identifiers  $\mathcal{O}$ , names  $\mathcal{N}$ , and atomic values  $\mathcal{V}$ . An entity is a design artifact used to model a real-world object. It consists of a unique entity identifier and a set of attributes. An *attribute* is a pair  $\langle n, v \rangle$  of a name and a value and describes some characteristic of the entity. The set  $\mathcal{A} = \mathcal{N} \times \mathcal{V}$  represents the infinite set of all the possible attributes.

**Definition 9.1.** An *entity*  $e$  is a tuple  $\langle id, A \rangle$  called the *entity identifier* of the entity and  $A \subseteq \mathcal{A}$  is a finite set called the set of *entity attributes*.

■ 290

Since each entity is distinguished by its unique identifier, for the rest of the document, the term entity and entity identifier will be used interchangeably.

291  
292

Entity identifiers can be considered a special type of atomic values, allowing 293  
identifiers to serve as the value of an attribute. Through this mechanism, the 294  
data model is able to support not only entities and their characteristics but also 295  
relationships among them. 296

A database is a set of entities. Among these entities, there may be groups 297  
modeling the same real-world object but using a different or overlapping set of 298  
attributes. Such entities are said to be *linked*. 299

**Definition 9.2.** A *database* is a tuple  $\langle \mathcal{E}, \mathcal{L} \rangle$ , where  $\mathcal{E}$  is a finite set of entities and  $\mathcal{L}$  300  
is a linkage assignment on  $\mathcal{E}$ . A *linkage assignment* over a set  $E$  is a binary relation 301  
 $L \subseteq E \times E$  that is commutative, symmetric, and reflexive. Two entities  $e_1, e_2 \in \mathcal{E}$  302  
of a database  $\langle \mathcal{E}, \mathcal{L} \rangle$  are said to be *linked* and is denoted as  $e_1 \equiv e_2$  if  $(e_1, e_2) \in \mathcal{L}$ . 303  
A maximal group of entities that are pairwise linked forms a *factor*. ■ 304

It is important to note that a linkage assignment can be equivalently expressed 305  
either through an explicit statement of the binary relationships or through a set of 306  
groups of entities, with each such group representing a factor. For instance, given 307  
six entities  $e_1, e_2, \dots, e_6$ , the set  $\{\{e_1, e_2, e_3\}, \{e_4, e_5\}, \{e_6\}\}$  describes a linkage 308  
assignment with three factors. The first factor consists of entities  $e_1, e_2$ , and  $e_3$ ; the 309  
second contains  $e_4$  and  $e_5$ ; and the third contains only  $e_6$ . The set of linkages are the 310  
set of all the pairwise links in each factor. 311

Since linked entities in a database represent the same real-world object, they 312  
can be replaced by a new entity that combines the information described by them. 313  
(Note that linked entities may have different, or even disjoint, sets of attributes) 314  
This process is referred to as *entity merge* and leads to more compact database 315  
representations without losing any information. A database in which no merge can 316  
be performed is said to be *minimal*. 317

**Definition 9.3.** The *merge* of a set of entities  $e_i = \langle id_i, A_i \rangle$  for  $1 \leq i \leq n$ , denoted 318  
as  $\text{merge}(e_1, e_2, \dots, e_n)$ , is a new entity  $\langle id, A \rangle$  such that  $id$  is a new identifier and 319  
 $A = \cup_{i=1}^n A_i$ . The *minimal* form of a database  $\langle \mathcal{E}, \mathcal{L} \rangle$  is a database  $\langle \mathcal{E}', \mathcal{L}' \rangle$ , where 320  
 $\mathcal{L}' = \emptyset$  and  $\mathcal{E}' = \{e | e = \text{merge}(e_1, e_2, \dots, e_n) \wedge \{e_1, e_2, \dots, e_n\} \text{ is a factor in } 321$   
 $\langle \mathcal{E}, \mathcal{L} \rangle\}$ . ■ 322

To capture the uncertainty that may exist on the data, every attribute of an entity 323  
is associated with a value between 0 and 1, which indicates a likelihood that the 324  
information described by the attribute is among the characteristics of the real-world 325  
object that the entity models. Uncertainty exists also on the linkage information 326  
among the entities. Thus, we extend the definition of the database to include this 327  
uncertainty. 328

**Definition 9.4.** A *probabilistic linkage database* is a tuple  $\langle \mathcal{E}, \mathcal{L}, p^a, p^l \rangle$ , where 329  
 $\mathcal{E}$  is a set of entities and  $\mathcal{L}$  is a linkage assignment on  $\mathcal{E}$ .  $p^a$  is a function that 330  
assigns a probability weight to the attributes of the entities, i.e.,  $p^a | B \mapsto [0, 1]$  with 331  
 $B = \{a | a \in A \wedge \langle id, A \rangle \in \mathcal{E}\}$ .  $p^l$  is also a function that assigns a probability weight 332  
to the entity linkages, i.e.,  $p^l | \mathcal{L} \mapsto [0, 1]$ . ■ 333

Note, that our notion of a probabilistic database goes beyond the traditional 334  
probabilistic database [12] that simply associates probabilities with attributes, by 335  
assigning probabilities also to linkage relationships that exist among the entities. 336

*Example 9.1.* Figure 337 illustrates a small fraction of a probabilistic linkage 338  
database. It contains a total of five entities; therefore,  $\mathcal{E} = \{e_1, e_2, e_3, e_4, e_5\}$ . The 339  
entity linkage techniques we executed on these five entities generated three entity 340  
linkages, and thus,  $\mathcal{L} = \{l_{e_1,e_2}, l_{e_1,e_3}, l_{e_4,e_5}\}$ . From the figure, we can also see the 341  
attributes composing the entities. For example, entity  $e_1$  has five attributes, with 342  
the first attribute having a value “Harry Potter and the Chamber of Secrets” for the 343  
name “title”. The probability for the specific attribute is 0.6, which corresponds to 344  
the belief we have that the specific attribute describes  $e_1$ . These probabilities are 345  
provided by function  $p^a$ . Similarly, function  $p^l$  provides the probabilities for the 346  
entity linkages, shown in the figure with the line between the entities. ■ 346

Having the right probabilities on the attributes of a probabilistic entity is a 347  
critical issue. One of the challenges is to understand the semantics of these 348  
probability numbers and to adequately use them to perform the merge. This task is 349  
challenging since very often, different algorithms may have been used to compute 350  
the probabilities for the attributes of different entities, or the employed algorithms 351  
may not be known. In certain cases, these numbers may not even be probabilities in 352  
the strict mathematical sense but rather numbers that are meant to provide a relative 353  
ranking of the likelihood of the respective attributes in the entity. This can make 354  
the computation of the query results even harder. The same issues arise on the 355  
linkage information. There is a large amount of literature on the topic of computing 356  
entity linkage [19], with most methods analyzing the structural similarity of the 357  
data and returning some numbers measuring the likelihood that two data structures 358  
represent the same real-world entity. All these issues are outside of the scope of the 359  
current paper. We assume that this information is explicitly provided or computed 360  
in advance using some data analysis tools [6]. ■ 361

A probabilistic linkage database models multiple different real-world situations, 362  
i.e., possible databases, depending on what entity linkages actually exist among the 363  
entities and on what attributes each entity actually has. To differentiate between the 364  
situations that arise from the different linkages, the notion of *possible l-worlds* is 365  
introduced (short for possible linkage worlds). ■ 366

**Definition 9.5.** Given a probabilistic linkage database  $\langle \mathcal{E}, \mathcal{L}, p^a, p^l \rangle$ , a *consistent* 367  
linkage specification is a linkage assignment  $\mathcal{L}^{sp}$  such that  $\forall x, y \in \mathcal{L}^{sp}: x, y \in \mathcal{L} \wedge$  368  
 $p^l(x, y) = 0$ . The probabilistic database with linkages  $\langle \mathcal{E}, \mathcal{L}^{sp}, p^a, p_{sp}^l \rangle$ , where 369  
 $p_{sp}^l(x, y) = 1, \forall (x, y) \in \mathcal{L}^{sp}$ , is called a *possible l-world*. The set of all the 370  
possible l-worlds of probabilistic database with linkages  $\langle \mathcal{E}, \mathcal{L}, p^a, p^l \rangle$  is denoted 371  
as  $plw(\langle \mathcal{E}, \mathcal{L}, p^a, p^l \rangle)$ . ■ 372

Since a linkage specification uniquely defines a possible l-world (Definition 9.5), 373  
the terms “consistent linkage specification” and “possible l-world” will be used 374  
equivalently. Furthermore, we will consider only consistent linkage specifications; 375  
thus, we will not mention the word “consistent” any more. ■ 376

**Fig. 9.3** An illustration of the linkage specification  $\mathcal{L}^{\text{sp}} = \{l_{e_1,e_2}, l_{e_4,e_5}\}$  for the probabilistic linkage database shown in Fig. 9.2

• $e_{12}$	<table style="width: 100%; border-collapse: collapse;"> <tr><td>title:</td><td>Harry Potter and the Chamber of Secrets</td><td>0.6</td></tr> <tr><td>starring:</td><td>Daniel Radcliffe</td><td>0.7</td></tr> <tr><td>starring:</td><td>Emma Watson</td><td>0.4</td></tr> <tr><td>writer:</td><td>J.K. Rowling</td><td>0.6</td></tr> <tr><td>genre:</td><td>Fantasy</td><td>0.6</td></tr> <tr><td>title:</td><td>Harry Potter and the Chamber of Secrets</td><td>0.7</td></tr> <tr><td>date:</td><td>2002</td><td>0.8</td></tr> <tr><td>starring:</td><td>Daniel Radcliffe</td><td>0.5</td></tr> <tr><td>starring:</td><td>Emma Watson</td><td>0.9</td></tr> </table>	title:	Harry Potter and the Chamber of Secrets	0.6	starring:	Daniel Radcliffe	0.7	starring:	Emma Watson	0.4	writer:	J.K. Rowling	0.6	genre:	Fantasy	0.6	title:	Harry Potter and the Chamber of Secrets	0.7	date:	2002	0.8	starring:	Daniel Radcliffe	0.5	starring:	Emma Watson	0.9	35.1
title:	Harry Potter and the Chamber of Secrets	0.6																											
starring:	Daniel Radcliffe	0.7																											
starring:	Emma Watson	0.4																											
writer:	J.K. Rowling	0.6																											
genre:	Fantasy	0.6																											
title:	Harry Potter and the Chamber of Secrets	0.7																											
date:	2002	0.8																											
starring:	Daniel Radcliffe	0.5																											
starring:	Emma Watson	0.9																											
• $e_3$	<table style="width: 100%; border-collapse: collapse;"> <tr><td>title:</td><td>Harry Potter and the Chamber of Secrets</td><td>0.8</td></tr> <tr><td>genre:</td><td>Fantasy</td><td>0.8</td></tr> <tr><td>writer:</td><td>J.K. Rowling</td><td>0.7</td></tr> </table>	title:	Harry Potter and the Chamber of Secrets	0.8	genre:	Fantasy	0.8	writer:	J.K. Rowling	0.7	35.2																		
title:	Harry Potter and the Chamber of Secrets	0.8																											
genre:	Fantasy	0.8																											
writer:	J.K. Rowling	0.7																											
• $e_{45}$	<table style="width: 100%; border-collapse: collapse;"> <tr><td>codename:</td><td>The Big Blue</td><td>0.8</td></tr> <tr><td>location:</td><td>California</td><td>0.5</td></tr> <tr><td>name:</td><td>International Business Machines</td><td>0.9</td></tr> <tr><td>base:</td><td>New York</td><td>0.7</td></tr> <tr><td>date:</td><td>2002</td><td>0.7</td></tr> </table>	codename:	The Big Blue	0.8	location:	California	0.5	name:	International Business Machines	0.9	base:	New York	0.7	date:	2002	0.7	35.3												
codename:	The Big Blue	0.8																											
location:	California	0.5																											
name:	International Business Machines	0.9																											
base:	New York	0.7																											
date:	2002	0.7																											

**Table 9.1** A summary of the notation introduced in Sect. 9.3 and used throughout this chapter

Notation	Description	
$a_i = \langle n, v \rangle$	Attribute: a pair of name $n$ and value $v$	t35.1
$e_i = \langle id, A \rangle$	Entity: a tuple with an identifier $id$ and a set of attributes $A$	t35.2
$l_{e_i, e_j}$	Linkage: denotes a possible match between entity $e_i$ with entity $e_j$	t35.3
$\langle \mathcal{E}, \mathcal{L}, p^a, p^l \rangle$	Probabilistic linkage database	t35.4
$\text{plw}(\langle \mathcal{E}, \mathcal{L}, p^a, p^l \rangle)$	Possible l-worlds of a probabilistic linkage database	t35.5
$\mathcal{L}^{\text{sp}}$	Linkage assignment	t35.6
$f_i$	Factor: a set pairwise linked entities	t35.7
		t35.8

*Example 9.2.* Consider again the probabilistic linkage database of Fig. 9.2. The entity linkage set is  $\mathcal{L} = \{l_{e_1,e_2}, l_{e_1,e_3}, l_{e_4,e_5}\}$ . One of the possible linkage specifications  $\mathcal{L}^{\text{sp}} = \{l_{e_1,e_2}, l_{e_4,e_5}\}$ , which means accepting the two out of the three linkages of this original entity linkage set  $\mathcal{L}$ . As explained, each linkage we accept implies a merge between the entities of the linkages. Therefore, the specific linkage specification means we need to merge  $e_1$  with  $e_2$  and  $e_4$  with  $e_5$ . The result is a probabilistic database with linkages, as shown in Fig. 9.3. ■ 383

In the remaining chapter, we will use the notation  $l_{e_1,e_2}$  as a shorthand for  $(e_1, e_2) \in \mathcal{L}$ . A summary of the introduced notation, which is used throughout this chapter, is listed in Table 9.1. 384  
385  
386

Note that a possible l-world still has probabilities assigned to the attributes of its elements. The possible worlds of each possible l-world describe a number of non-probabilistic databases, depending on whether each probabilistic attribute is present or not. These nonprobabilistic databases are referred to as *regular database*, or a *solution*. A solution actually corresponds to what is referred to as a possible world in the probabilistic database literature [12]. 387  
388  
389  
390  
391  
392

For queries, a simple, but flexible, query language is adopted. A query is a series 393  
of attributes, i.e., a list of name–value pairs. Intuitively, the semantics of the query is 394  
to discover entities that contain the attributes described in its attribute list. An entity 395  
 $e$  is included in the answer of a query  $q$  if it contains all the requested attributes. 396  
Evaluating the query on a probabilistic linkage database can be performed on the 397  
basis of the traditional query answering approaches. 398

Using entity linkage, it is possible to go beyond this expressiveness and be able 399  
to retrieve new entities that may not be explicitly represented in the database. These 400  
entities result from the merge of two or more entities as specified by their linkage 401  
information. In particular, the results of evaluating a query  $q$  over a probabilistic 402  
linkage database with a linkage  $\mathcal{L}$  is equal to the results of the evaluation of the query 403  
 $q$  over the minimal form of the database, as specified by the linkage assignment  $\mathcal{L}$ . 404

If the database is probabilistic, and a specific linkage assignment  $\mathcal{L}$  has been 405  
decided, then its minimal form can be computed but will also be probabilistic, 406  
since the entity attributes will have probabilities. Evaluation of a query over such 407  
a database can be performed based on various techniques proposed for probabilistic 408  
databases [12]. However, if the linkage assignment is also probabilistic, as in the 409  
general case of a probabilistic database, then a query is evaluated over the database 410  
by computing all its possible worlds, i.e., all the possible linkage assignments, and 411  
then evaluating the query on the minimal form of each such worlds. The final result 412  
of the query will be the union of the results of the evaluations on the individual 413  
worlds. However, since the linkage information is probabilistic in the first place, so 414  
is the linkage assignment, and as a consequence, the results of each evaluation on 415  
the individual worlds should also be coming with some probability. 416

A fundamental property that differentiates our work from other work related 417  
to querying probabilistic data is that the query results are computed from entities 418  
that are compiled on the fly at query execution time from the available linkage 419  
information. The entities used in query evaluation thus consider entities that are 420  
not materialized in the repository in this form. In particular, traditional approaches 421  
evaluate the queries on the extensional data that can be found in the database. In 422  
our work, we view the probabilistic entity linkage as an intensional description of 423  
a number of possible entities that can result from the merge of those linked. We 424  
compute these entities through the possible worlds on the fly and offer additional 425  
query results. 426

*Example 9.3.* Consider again the probabilistic linkage dataset illustrated in Fig. 9.2 427  
and the query: 428

(starring: “Emma Watson”, starring: “Radcliffe, Daniel” ) 429

In can be observed that there is no entity with both the attributes requested by the 430  
query. Traditional query answering techniques would have failed to return results 431  
or would have returned with a low confidence, those having at least one of the two 432  
requested attributes. However, in the possible world described by the entity linkage 433  
 $\{\{e_1, e_2, e_3, e_4, e_6\}, \{e_8, e_9\}\}$ , the merge of the entities of the first of the two factors 434  
represents an entity with both the attributes requested in the query. Thus, such an 435  
entity can be returned with a much higher confidence. ■ 436

The following sections deal with the challenge of computing the right probabilities of the possible l-worlds and possible worlds, and performing query answering on the fly without having to materialize all them. 437  
438  
439

## 9.4 Efficient Query Evaluation

440

In this section, we present how query evaluation can be performed efficiently over a probabilistic linkage database. A more detailed description of the algorithm and experimental evaluation is available in [24] and [25]. 441  
442  
443

Our query evaluation approach is based on an idea similar to the one incorporated in probabilistic databases for dealing with datasets of large sizes. In particular, the probability values on the linkages are interpreted as the probability distribution over the set of all the possible l-worlds  $plw(\langle \mathcal{E}, \mathcal{L}, p^a, p^l \rangle)$ . Given a linkage specification, i.e., a possible l-world, the probability values of the attributes are interpreted as the probability distribution over all the possible solutions (ref. Sect. 9.3) within one possible world leading to a two-level approach. The answer of a query on a probabilistic linkage database, thus, is the union of the answers over all the possible worlds of all the l-worlds fulfilling the query conditions. Each element in the answer set, however, is accompanied by an aggregated probability value which reflects the probability of existence of the specific possible world, which contains the answer element, as well as the probability of the possible l-world, which contains the respective solution. 444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456

This description of semantics for query answering is indirectly suggesting a query evaluation strategy: compute all possible worlds of all possible l-worlds, compute their probability of existence, evaluate the query in each one of them, and return the results along with computed probability. Unfortunately, following this approach is prohibitively expensive. 457  
458  
459  
460  
461

Here, we propose an alternative evaluation strategy that avoids the high computational cost without any loss of accuracy. The basic idea is to restrict the computation to only those possible l-worlds that are meaningful for the query at hand. Since there is a one-to-one correspondence between possible l-worlds and linkage specifications, and between linkage specifications and entity merges, we start from the entity merges that are required in order to generate an answer to the given query. From the merges, we can find the linkage specifications, and from these assignments, the possible l-worlds. The probability of each possible l-world is computed based on the probabilities of the linkages included or not included when generating the specific l-world. Finally, the possible worlds of each l-world are generated along with their own probability, which is combined with the probability of the respective world and then included in the answer set of the query. An overview of the proposed query evaluation is given by Algorithm 18, while the following subsections describe these steps of the algorithm in more detail. 462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475

**Algorithm 18:** Query evaluation

---

**Input:**  $Q$ : a query describing an entity  
**Output:**  $R$ : a set of entities satisfying query conditions  
 $LS \leftarrow \text{findRequiredLinkageSpecifications}(Q);$   
 $PLW \leftarrow \emptyset;$   
 $R \leftarrow \emptyset;$   
**foreach**  $ls \in LS$  **do**  
   $W \leftarrow \text{findPossibleLWorlds}(ls);$   
  **foreach**  $w \in W$  **do**  
     $w.\text{prob} \leftarrow \text{calculateWorldLProbability}(ls);$   
     $PLW \leftarrow PLW \cup \{w\};$   
  **foreach**  $plw \in PLW$  **do**  
     $E \leftarrow \text{evaluateQuery}(plw, Q);$   
    **foreach**  $e \in E$  **do**  
       $e.\text{prob} \leftarrow \text{combineProb}(e.\text{prob}, plw.\text{prob});$   
     $R \leftarrow R \cup \{e\};$

---

**9.4.1 Indexing Structure**

476

A commonly used approach in answering queries over probabilistic data is to 477 partition the data into a series of disjoint/independent groups [4, 12, 32, 34]. These 478 groups can be found in the literature under the name factors [34] or components [4]. 479 The set of the possible combinations between the data of these groups produces all 480 the possible worlds. 481

This idea is not directly applicable to our case. The reason is that the existing 482 approaches operate under the assumption that the data within one factor or compo- 483 nent are independent. This assumption does not hold in our case. The transitive 484 property of the linkages may generate additional correlation, i.e., dependencies, 485 that are equally important for the correct identification of the possible worlds. For 486 instance, entity linkages  $l_{e_1,e_2}$  and  $l_{e_1,e_3}$  without linkage  $l_{e_2,e_3}$  cannot be considered, 487 since the first two linkages imply the information encoded in the third linkage. 488

We follow an idea similar to the management of uncertain data with correlations 489 [34]. As a first step, we divide the set of entities into sets of connected components, 490 i.e., factors (Definition 9.2). For example, given  $\mathcal{L} = \{l_{e_1,e_2}, l_{e_1,e_3}, l_{e_2,e_3}, l_{e_8,e_9}\}$ , two 491 independent factors can be identified. The first factor contains entities  $e_1, e_2$ , and  $e_3$  492 with linkages  $L_1 = \{l_{e_1,e_2}, l_{e_1,e_3}, l_{e_2,e_3}\}$ , while the second factor contains simply  $e_8$  493 and  $e_9$  with one linkage  $L_2 = \{l_{e_8,e_9}\}$ . 494

To compute all possible l-worlds of a probabilistic linkage database  $\langle \mathcal{E}, \mathcal{L}, p^a, p^l \rangle$ , 495 we need to consider all the possible valid linkage specifications of  $\mathcal{L}$ . This number 496 can easily get large enough to make the computation intractable. Based on the 497 fact that no linkage exists between entities in different factors, we can improve 498 the situation by considering each factor independently. We will use notation  $\mathcal{L}_{f_i}^{\text{sp}}$  to 499 denote all the possible valid linkage specifications between entities of the  $i$ th factor 500 and  $\mathcal{L}_{f_i}^{\text{sp}}(k)$  one of the possible assignments. 501

Each possible l-world is using a specific specification within each factor. Thus, 502  
 the set of possible l-worlds can be derived by combining all the alternative valid 503  
 linkage specifications within each factor as follows: 504

$$\text{plw}(\langle \mathcal{E}, \mathcal{L}, p^a, p^l \rangle) = \mathcal{L}_{f_1}^{\text{sp}} \times \mathcal{L}_{f_2}^{\text{sp}} \times \cdots \times \mathcal{L}_{f_n}^{\text{sp}}$$

*Example 9.4.* Consider a probabilistic linkage database with entity linkages  $\mathcal{L} = \{l_{e_1,e_2}, l_{e_1,e_3}, l_{e_4,e_5}\}$ . For the specific set of entity linkages, we have two factors. Factor 505  
 $f_1$  with entities  $\{e_1, e_2, e_3\}$  and factor  $f_2$  with entities  $\{e_4, e_5\}$ . The corresponding 506  
 linkage specifications are  $\mathcal{L}_{f_1}^{\text{sp}} = \{l_{e_1,e_2}, l_{e_1,e_3}\}$  and  $\mathcal{L}_{f_2}^{\text{sp}} = \{l_{e_4,e_5}\}$ , and thus, the 507  
 possible l-worlds can be retrieved as follows: 508

$$\begin{array}{ll} \mathcal{L}_{f_1}^{\text{sp}}(1) = \{l_{e_1,e_2}, l_{e_1,e_3}\} & \mathcal{L}_{f_2}^{\text{sp}}(1) = \{l_{e_4,e_5}\} \\ \mathcal{L}_{f_1}^{\text{sp}}(2) = \{l_{e_1,e_2}\} & \times \quad \mathcal{L}_{f_2}^{\text{sp}}(2) = \{\} \\ \mathcal{L}_{f_1}^{\text{sp}}(3) = \{l_{e_1,e_3}\} & \\ \mathcal{L}_{f_1}^{\text{sp}}(4) = \{\} & \end{array} \quad 509$$

The cartesian product of these linkage assignments results into the number of 511  
 possible l-worlds for the whole database. The next table illustrates these l-worlds 512  
 (through the specifications) along with the entity merges for each of these possible 513  
 l-world. 514

Possible worlds	Entity merges
$D_1 = \{l_{e_1,e_2}, l_{e_1,e_3}, l_{e_4,e_5}\}$	$e_1 \equiv e_2 \equiv e_3, e_8 \equiv e_9$
$D_2 = \{l_{e_1,e_2}, l_{e_1,e_5}\}$	$e_1 \equiv e_2 \equiv e_3, e_8, e_9$
$D_3 = \{l_{e_1,e_2}, l_{e_4,e_5}\}$	$e_1 \equiv e_2, e_3, e_8 \equiv e_9$
$D_4 = \{l_{e_1,e_2}\}$	$e_1 \equiv e_2, e_3, e_4, e_5$
$D_5 = \{l_{e_1,e_3}, l_{e_4,e_5}\}$	$e_2, e_1 \equiv e_3, e_4 \equiv e_5$
$D_6 = \{l_{e_1,e_3}\}$	$e_2, e_1 \equiv e_3, e_4, e_5$
$D_7 = \{l_{e_4,e_5}\}$	$e_1, e_2, e_3, e_4 \equiv e_5$
$D_8 = \{\}$	$e_1, e_2, e_3, e_4, e_5$

■ 515

To avoid recomputing the factors every time, we maintain an index structure 516  
 which is dynamically maintained. The index structure is based on the idea of equiv- 517  
 alence classes. In reality, each factor is actually an equivalence class. When data are 518  
 modified and new linkages are introduced or old ones are eliminated, changes should 519  
 occur in the equivalence class memberships, thus on the factors. Algorithm 19 520  
 illustrates how the factor index is maintained under new linkage insertions. 521

Once the various l-worlds have been constructed, the second important step is to 522  
 compute the probability of each possible l-world. Since the factors are independent 523  
 of each other, the probability of an l-world can be computed as the product of the 524  
 probabilities of the involved factors (Sect. 9.4.3). It is thus given by: 525

**Algorithm 19:** Updating factor indexing

---

**Input:** (a)  $l_{\alpha, \beta} := (e_\alpha, e_\beta, p)$ : a new linkage  
 (b)  $F := F_1, F_2, \dots, F_n$ : the factors

**Output:** Updated factors  $F$

```

 $F_\alpha \leftarrow \text{getFactorOf}(e_\alpha);$ 
 $F_\beta \leftarrow \text{getFactorOf}(e_\beta);$ 
if  $F_\alpha \notin F$  then
|    $F \leftarrow F \cup F_\alpha;$ 
if  $F_\beta \notin F$  then
|    $F \leftarrow F \cup F_\beta;$ 
if  $F_\alpha == F_\beta$  then
|   // already in the same factor ;
|   return  $F$ ;
else if  $F_\alpha != F_\beta$  then
|    $F_\gamma \leftarrow F_\alpha \cup F_\beta;$ 
|    $F \leftarrow F \setminus F_\alpha \setminus F_\beta \cup F_\gamma;$ 

```

---

$$\Pr(l - \text{world}) = \prod_{i=1}^n \Pr(\mathcal{L}_{f_i}^{\text{sp}}(.)) \quad (9.1)$$

*Example 9.5.* Assume that the possible worlds that satisfy condition  $e_1 \equiv e_2$  and  $e_4 \equiv e_5$  needs to be retrieved. Based on Example 9.4, it can be easily seen that the left part of the condition is satisfied by factors  $\mathcal{L}_{f_1}^{\text{sp}}(1)$  and  $\mathcal{L}_{f_1}^{\text{sp}}(3)$ , while the right part by factor  $\mathcal{L}_{f_2}^{\text{sp}}(1)$ . Therefore, only the possible l-worlds given by the product of these factors needs to be considered. These are: 530

$$\begin{aligned} \mathcal{L}_{f_1}^{\text{sp}}(1) &= \{l_{e_1, e_2}, l_{e_1, e_3}\} & \times & \mathcal{L}_{f_2}^{\text{sp}}(1) = \{l_{e_4, e_5}\} \\ \mathcal{L}_{f_1}^{\text{sp}}(3) &= \{l_{e_1, e_3}\} \end{aligned} \quad 531$$

This results in two possible l-worlds. The first l-world is given by 532

$$\text{plw}_1 = \mathcal{L}_{f_1}^{\text{sp}}(1) \times \mathcal{L}_{f_2}^{\text{sp}}(1) \text{ with probability } \Pr(\mathcal{L}_{f_1}^{\text{sp}}(1)) \cdot \Pr(\mathcal{L}_{f_2}^{\text{sp}}(1)) \quad (9.2)$$

and the second l-world is given by 533

$$\text{plw}_2 = \mathcal{L}_{f_1}^{\text{sp}}(3) \times \mathcal{L}_{f_2}^{\text{sp}}(1) \text{ with probability } \Pr(\mathcal{L}_{f_1}^{\text{sp}}(3)) \cdot \Pr(\mathcal{L}_{f_2}^{\text{sp}}(1)) \quad 534$$

■ 535

#### 9.4.2 Retrieving Possible l-Worlds

536

In order to avoid the creation of all possible l-worlds when a query is issued, we exploit the list of factors we are maintaining, as mentioned in the previous 537 538

section, and the attributes in the query. We do so in order to restrict the creation of only those possible l-worlds that are necessary, i.e., those that will lead to the generation of some query results. We achieve this by first detecting the entity merges that are required in order to satisfy the conditions of the query. In particular, for every attribute specification  $a_i$  in the query, a list  $E_{a_i}$  is constructed with all the entities having attribute  $a_i$ . Clearly an entity satisfies all the query conditions, that is, contains all the attributes set by the query, if it appears in each one of the lists  $E_{a_i}$ , for  $i = 1..n$ . It is not always the case that such an entity exists. However, by merging two or more entities, it is possible to create a new one (the result of the merge) whose attributes contain all the attributes in the query. Let  $S$  be a set of such entities. For the set  $S$  to serve the required purpose, two main properties need to be satisfied. First, all its members have to belong to the same factor (it is not possible to talk about merge of entities that belong to different factors), and second, there must be at least one entity  $e \in S$  from every list  $E_{a_i}$ , with  $i = 1..n$ . Of course, one can always create “super entities” by merging all the entities in every factor. However, this may generate merges that are not necessary. To avoid this form of redundancy, we require that each set  $E_{a_i}$  contributes at most one entity. This means that the number of entities to merge can never be more than the number of attributes in the query.

To compute all the possible merge combinations, we generate the cartesian product of the sets  $E_{a_i}$  with the extra requirement that they should belong to the same factor. Algorithm 20 provides a brief description of the described steps.

*Example 9.6.* For example, assume that a query  $q$  contains attributes  $a_1, a_2$ , and  $a_3$ , and each one is satisfied by the entity sets  $E_{a_1} = \{f_1 - e_a, f_1 - e_b, f_2 - e_c\}$ ,  $E_{a_2} = \{f_1 - e_b, f_2 - e_d\}$ , and  $E_{a_3} = \{f_1 - e_g, f_1 - e_h, f_2 - e_i\}$ , respectively. For each entity in the sets, we also indicate the factor in which the entity belongs to. We first compute all the possible combinations:

$E_{a_1}$	$E_{a_2}$	$E_{a_3}$
$f_1 - e_a$	$\times_{(E_{a_1}, f_i = E_{a_2}, f_j)}$	$f_1 - e_g$
$f_1 - e_b$	$f_1 - e_b$	$f_1 - e_h$
$f_2 - e_c$	$f_2 - e_d$	$f_2 - e_i$

which lead to the following merges:  $\text{merge}(e_a, e_b, e_g)$ ,  $\text{merge}(e_a, e_b, e_h)$ ,  $\text{merge}(e_b, e_g)$ ,  $\text{merge}(e_b, e_h)$ , and  $\text{merge}(e_c, e_d, e_i)$ . Note that  $e_b$  belongs to both sets  $E_{a_1}$  and  $E_{a_2}$ , which means that the entity  $e_b$  has both attributes  $a_1$  and  $a_2$ ; thus, being merged with  $e_g$  or  $e_h$  is enough to create an entity that satisfies the query conditions. ■

Consider now a merge of entities  $e_\alpha$  and  $e_\beta$  from a factor  $f_i$ . From all possible worlds that can be generated from the specific factor, we are only interested in those that contain the specific entity merge. The remaining can be ignored. All entities constructed from this merge will contain some common attributes, which correspond to the attributes directly coming from the entities  $e_\alpha$  and  $e_\beta$ . The

**Algorithm 20:** Generate entity merges

---

**Input:**  $Q := \langle a_1, a_2, \dots, a_k \rangle$ : an entity query  
 $\langle \mathcal{E}, \mathcal{L}, p^a, p^l \rangle$   
: a probabilistic linkage database **Output:**  $M$ : a set of entity merges

**foreach**  $a_i$  in  $Q$  **do**

$$\quad \bigcup E_i \leftarrow \{e \mid e = \langle id, A \rangle \wedge e \in \mathcal{E} \wedge a_i \in A\};$$

$$\quad N \leftarrow \{(e_1, \dots, e_n) \mid \forall i = 1..n : e_i \in E_i \wedge \forall i = 2..n : factor(e_{i-1}) = factor(e_i)\};$$

$$\quad M \leftarrow \{\text{eliminateDuplicates}(m) \mid m \in N\};$$


---

remaining attributes in the possible l-worlds will come from entities participating in other linkages which exist in the specific possible l-world. To further optimize our framework, we exploit this behavior and instead of returning all the entities for each factor, we return only one *partial entity*. The partial entity contains the minimum set of attributes required by the specific merge since this is commonly found in all entities generated by this factor. 577  
578  
579  
580  
581  
582

#### 9.4.3 Computing Probabilities of Possible l-Worlds

583

The next step in the query answering process is to decide the probability of a merge, i.e., a partial match as mentioned in the previous section. Recall that a partial match may be true in many possible worlds. There are two alternatives that one can follow. The first is to assign as the probability of the partial match the sum of the probabilities of these worlds. The second is to compute and consider only the maximum of these probabilities. The latter requires significantly less computation time, since it only needs to identify the world with the highest probability. For systems that simply use the match probability as a ranking mechanism for the entities before displaying them to the user, this second option is typically sufficient. 584  
585  
586  
587  
588  
589  
590  
591  
592

The algorithm for computing the match probability is based on the algorithm for finding shortest paths on graphs. In particular, provided the entity linkages  $L_i$  in a factor, we generate a weighted undirected graph  $G$  as follows: every entity participating in the linkages of  $L_i$  becomes a node of the graph. Each linkage  $l_{e_\alpha, e_\beta}$  becomes an edge that connects the nodes representing entities  $e_\alpha$  and  $e_\beta$ . The weight of such an edge is given by the probability of the respective linkage. 593  
594  
595  
596  
597  
598

An entity merging  $\text{merge}(e_1, e_2, \dots, e_n)$  corresponds to a spanning tree that connects all entities  $e_1, e_2, \dots, e_n$ . Computing the merge that maximizes the probability is similar to computing the maximum connected component of the graph that has the highest total probability (i.e., multiplication of the probabilities of its edges). Since the nodes of the graph correspond to the entities of a factor, they are all connected; thus, the maximum connected component will include all the nodes of the graph. 601  
602  
603  
604  
605  
Initially, all the entities (i.e., nodes) are marked as not visited. The highest ranked 606 edge is first selected and the two nodes it connects are marked as visited. Then a 607

list of edges is considered the subset of the edges that have one endpoint marked visited and one nonvisited. The one with the highest probability is selected and its nonvisited endpoint is marked as visited. The same step is repeatedly executed until all the nodes in the graph have been marked as visited. The probability of the merge is the multiplication of the probabilities of the edges that have been used in this process, and this probability is actually the maximum. 608  
609  
610  
611  
612  
613

#### 9.4.4 Retrieving and Computing Probabilities of Possible Worlds

614  
615

The previous sections have dealt with the problem of processing a query by 616 efficiently deriving the possible 1-worlds  $\text{plw}(\langle \mathcal{E}, \mathcal{L}_c, p^a, p^l \rangle)$  along with the cor- 617 responding entity merges. However, this is not all. Recall that entity attributes 618 themselves have probabilities; thus, even a possible 1-world may be describing 619 multiple different (nonprobabilistic) databases. These different databases are what 620 we call solutions. 621

Each solution essentially represents a different combination over the attributes 622 of the entities participating in a specific merge. For instance, consider the data 623 in Fig. 9.3 and, in particular, the attributes involved in  $\text{merge}(e_1, e_2)$ . The entity 624  $\text{merge}(e_1, e_2)$  needs to include all attributes from entities  $e_1$  and  $e_2$ , as shown in 625 Fig. 9.4. Two issues need to be taken into consideration. One is the probabilities 626 of the attributes, specifically in the case of duplication, and the other is the 627 dependencies that may exist among them. 628

The simplest approach regarding attribute dependencies is to consider the 629 attributes as independent and include them all as attributes in the merge result 630 entity. However, the attributes that appear in real-world datasets are not always 631 independent. The correlations (i.e., dependencies) between attributes that need to 632 be considered in attribute merge highly depend on the nature of the sources and 633 their datasets. Our framework is able to handle such correlations in a uniform 634 manner. A simple method is to cluster the exclusive attributes from each entity, i.e., 635

Attributes for $\text{merge}(e_1 \equiv e_2)$				Solutions			
aid.	name	value	p	(1)	(2)	(3)	(4)
• $a_{10}$	starring	Daniel Radcliffe	0.7	$a_{10}$	$a_{20}$	$a_{10}$	$a_{20}$
◊ $a_{11}$	starring	Emma Watson	0.4	$a_{11}$	$a_{11}$	$a_{21}$	$a_{21}$
$a_{12}$	writer	J.K. Rowling	0.6	$a_{12}$	$a_{12}$	$a_{12}$	$a_{12}$
$a_{13}$	genre	Fantasy	0.6	$a_{13}$	$a_{13}$	$a_{13}$	$a_{13}$
• $a_{20}$	starring	Radcliffe, Daniel	0.5				
◊ $a_{21}$	starring	Watson, Emma (II)	0.9				

**Fig. 9.4** The attributes involved in  $\text{merge}(e_1, e_2)$  along with the solutions when we deal with exclusive attributes (same name and similar values)

$M = \{\{e_1.\alpha_i, e_1.\alpha_j, \dots\}\}$ . We can then use this set to generate the solutions 636  
and compute their probability. The following paragraphs provide more details for 637  
generating solutions. 638

#### 9.4.4.1 Independent Attributes

639

One option is to assume no correlation between the attributes and thus no restrictions 640  
on which attributes to include in the merge result entity. In this case, there is only 641  
one solution which is given by the union of all entity attributes. 642

$$\text{merge}(e_1, e_2, \dots, e_n) = \langle id', \cup_{i=1}^n e_i.A \rangle$$

643

#### 9.4.4.2 Exclusive Attributes

644

In certain cases, the attributes originating from different entities participating in 645  
the entity merge are exclusive. This requires that only one occurrence of such an 646  
attribute to be in the entity resulted by the merge. A typical example of such an 647  
attribute is the distinct attribute names, e.g., a person can have only one name. 648  
Other examples are the attributes with the same name but similar (semantically or 649  
syntactically) values, e.g., attributes  $a_{11}$  and  $a_{21}$  from Fig. 9.4. A simple method is 650  
to cluster the exclusive attributes from each entity, i.e.,  $M = \{\{e_1.\alpha_i, e_1.\alpha_j, \dots\}\}$ . 651  
We can then use this set to generate worlds with these correlations: 652

$$\text{merge}(e_1, \dots, e_n) = \langle id', A \rangle, \text{ where}$$

$$A \subseteq (M_1 \times M_2 \times \dots \times M_m) \cup \{\alpha \mid \alpha \notin \cup_{i=1}^m M_i.\alpha\}.$$

The overall probability of a possible world depends on the probability of the 653  
attributes included or not included in the world. It is computed as the product of 654  
probability  $p^\alpha$  when attribute  $\alpha$  is part of the world and  $(1 - p^\alpha)$  when attribute  $\alpha$  655  
is not part of it: 656

$$\begin{aligned} \Pr(e' \mid \text{merge}(e_1, \dots, e_n)) &= \Pr(l - \text{world}) \times \prod_{\alpha \in e'.A} p^\alpha \\ &\quad \times \prod_{\alpha \notin e'.A \& \alpha \in e_i.A} (1 - p^\alpha). \end{aligned}$$

*Example 9.7.* Figure 9.4 shows the attributes involved in  $\text{merge}(e_1, e_2)$ . The exclusive 657  
attributes are given by set  $M = \{\{\alpha_{10}, \alpha_{20}\}, \{\alpha_{11}, \alpha_{21}\}\}$ . Figure 9.4 shows the 658  
four generated possible worlds, and their probability is computed according to above 659  
formula. 660

## 9.5 Evaluation

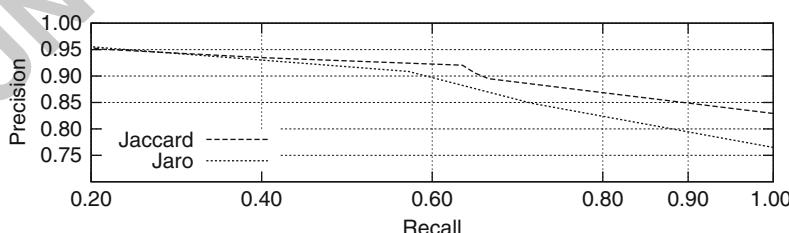
661

This section presents the results of the experimental evaluation for the suggested entity linkage methodology. The goal was twofold: (1) to study the effectiveness of our approach and identify its advantages over traditional techniques for entity linkage and (2) to investigate the efficiency of query processing and the overhead it introduces. We implemented our approach using Java 1.6 and performed all experiments on a computer with 5,400 rpm hard disk, a core 2 duo processor of 1.8 Ghz, and 2 GB RAM. For storing entities, we used MySQL 5, on the same computer. The following paragraphs present our datasets.

*Movie Dataset.* For evaluating the efficiency, we needed a sufficiently large dataset. Also, to investigate effectiveness, we needed linkages generated from different linkage techniques. We generated such a dataset by integrating data describing movies coming from two real-world systems, *IMDb* and *DBpedia*. *IMDb* data were stored in relational format, and *DBpedia* data were stored in RDF format. We converted both datasets to our data model and stored them in a relational database. To find the true matches (i.e., ground truth), we have used the *imdb\_id* field from the *DBpedia* dataset, which contains the id of the movie in the *IMDb* dataset. The table in Fig. 9.5 shows the details for the movie dataset.

For generating the entity linkages, we compared the movie titles using two standard string similarity methods [9], *Jaccard* and *Jaro*. Figure 9.5 plots the precision-recall plot resulting when using these techniques to link entities, with *Jaccard* being more successful in linking *IMDb* to *DBpedia* movies than *Jaro*. As expected, for both techniques, we see the clear dependency between precision and recall. While recall increases, precision decreases, and vice versa. The linkage techniques always have to decide the trade-off between precision and recall. In our experiments, we investigate how our approach addresses this issue.

	IMDb	DBpedia	Real-world objects
Entities:	23.182	28.040	13.435
Attributes:	820.999	186.655	



**Fig. 9.5** Information about the movie datasets: the data composing the dataset (*top*) and the precision-recall plot for two entity linkage techniques (*bottom*)

Attributes			Entity linkages (under threshold $t$ )					
Entities	Real-world objects		$t = 0.52$	$t = 0.58$	$t = 0.62$	$t = 0.68$	$t = 0.72$	$t = 0.78$
5764	2882	9774	12440	12012	10775	6394	5985	4184

**Fig. 9.6** Information about the Cora datasets: the data composing the dataset (*left*) and the number of linkages for various thresholds (*right*)

*Cora Dataset.* Our second dataset was a collection of publications and authors 687 from CiteSeer.<sup>1</sup> Cora dataset is typically used to evaluate entity linkage techniques 688 [3, 16, 17]. Its data comes from CiteSeer, a real-world application, and it contains 689 various author descriptions that refer to the same real-world object (maximum is 88 690 author instances describing the same real-world object). 691

We generated entity linkages between authors (i.e., entities) using the probabilistic 692 entity linkage [26]. Entity linkage techniques typically select a linkage threshold 693 and incorporate in the original data all entities with corresponding linkages above 694 this threshold. Figure 9.6 provides the details for this dataset along with the number 695 of linkages under different thresholds. Precision and recall of the generated entity 696 linkages are similar to the ones generated by other algorithms such as [16, 17]. For 697 our approach, we did not apply such a threshold but also used the linkages with low 698 probabilities. 699

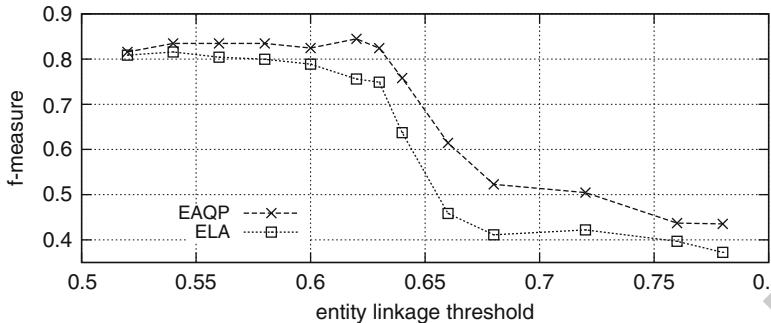
*Entity Queries.* The evaluations for both datasets were performed with 800 queries. 700 Each query was generated by randomly selecting attributes of entities belonging 701 to the same real-world object. We generated queries for real-world objects which 702 contained at least two entities in our dataset. All reported results are computed on 703 the average of 800 queries. 704

### 9.5.1 Effectiveness of Query Processing

We evaluate effectiveness of entity-aware query processing in a twofold manner. 706 First, we examine the quality of entities returned when querying with our approach. 707 Second, we compare entities returned from entity-aware query processing (EAQP) 708 with entities returned when we use directly the results of the entity linkage 709 techniques (ELA), i.e., applying the threshold on the entity linkages. We performed 710 both evaluations using the Cora Dataset. 711

Our first experiment was as follows. We used the entity linkage technique 712 to link all authors in the Cora dataset and stored the proposed linkages in the 713 database. Then we processed the 800 queries and compared the results returned with 714 EAQP and with ELA. As we already explained, entity linkage techniques select a 715 threshold and accept linkages that have a higher probability than this threshold. 716

<sup>1</sup><http://www.cs.umass.edu/~mccallum/data/cora-refs.tar.gz>



**Fig. 9.7** F-measure with entity-aware query processing (EAQP) and entity linkage algorithm (ELA) over various thresholds for the accepted linkages

Selecting a low threshold ( $t = 0.6$ ) will provide linkages with a high recall but 717 low precision, whereas selecting a higher threshold ( $t = 0.8$ ) will provide linkages 718 with significantly higher precision but also with lower recall. One can easily see 719 that when the selected threshold increases, the number of linkages is reduced since 720 we then accept only the entities with high probabilities (see Fig. 9.6 for the exact 721 numbers). We examine the behavior of the entity-aware query processing as well as 722 the entity linkage algorithm when the value of the threshold is increased. 723

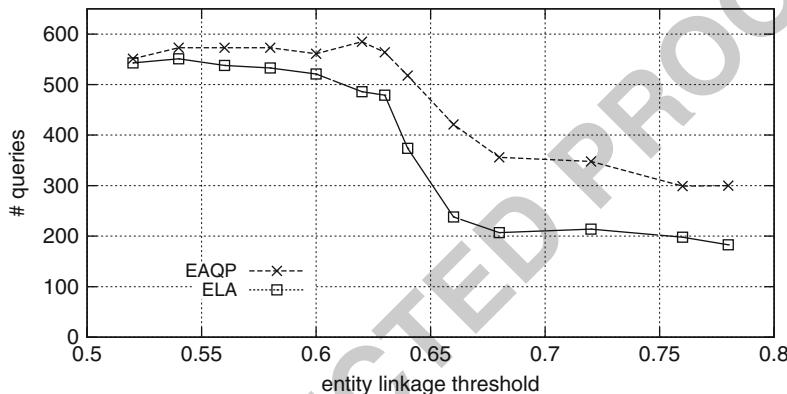
To measure quality of query results, we computed F-measure, which is a 724 weighted harmonic mean of precision and recall. We consider a query as correct 725 when it returns the real-world object by merging the information found in the 726 various corresponding entities. 727

Figure 9.7 shows the average F-measure of the 800 queries for various entity 728 linkage thresholds. As expected, when moving toward higher thresholds, the entity 729 linkage technique accepts less and less linkages. This makes the technique unable to 730 find the entities described by the queries. On the contrary, even for high thresholds, 731 EAQP is able to identify the entities. For example, for  $t = 0.66$ , EAQP returned the 732 correct entity for around 10% more queries than ELA. This is because EAQP can 733 find connected linkages to construct the entity described in the query. ELA had to 734 reject these linkages because of their low threshold. The exact precision and recall 735 values for some of these thresholds are shown in the following table: 736

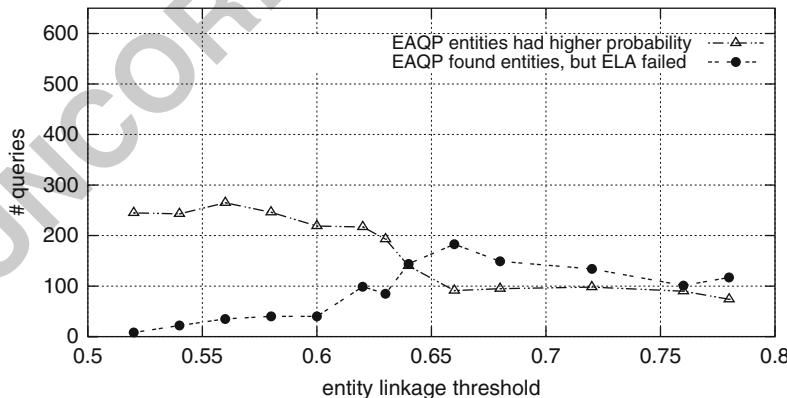
$t$	EAQP		ELA	
	Precision	Recall	Precision	Recall
0.62	1.00	0.73	1.00	0.61
0.63	0.99	0.71	1.00	0.60
0.64	0.91	0.65	1.00	0.47

Figure 9.8 shows the numbers of queries that were correctly answered for 737 different linkage thresholds. As shown, query processing with our approach returns 738 the correct results to more queries than ELA. 739

We further analyzed the results of this evaluation and identified two situations in which EAQP performs better than ELA. The first is that our approach has less failures, i.e., empty result set for queries. For instance, for  $t = 0.6$ , EAQP was able to return the correct answers for the 150 queries in which ELA did not return anything. The second situation is that there are cases in which the entities returned by EAQP were with higher confidence (i.e., with higher probability) than the entities returned by ELA. As shown in Fig. 9.8, for  $t = 0.6$ , EAQP returned 421 correct answers, whereas ELA returned 238 correct answers. For 91 answers, EAQP had higher probability than ELA. Figure 9.9 presents the numbers for these two situations.



**Fig. 9.8** Number of queries correctly returned with EAQP and ELA over various thresholds for the accepted linkages



**Fig. 9.9** Number of queries from the ones shown in Fig. 9.8 in which EAQP returned answers with higher probability and identified requested entities, whereas the ELA failed

### 9.5.2 Efficiency of Query Processing

750

We now report the results for the efficiency evaluation of our approach.

751

*Size of Generated Factors.* The core of our approach is based on generating factors by grouping linkages which are pairwise linked (Sect. 9.4.1). During query processing, we select the related factors and process them to construct the entities. The sizes of the factors influence the execution time of our approach.

752

753

754

755

We computed the size of the generated factors as the number of entity linkages contained in the factors. We then constructed the histogram of factor sizes. Figure 9.10 shows appearances per factor sizes as generated by both entity linkage techniques, *Jaro* and *Jaccard*. As shown, most of the factors have a small size and few factors are of bigger size. In addition, we see that for the entity linkage technique generated with *Jaro*, we have more factors of bigger sizes. Considering again the characteristics of the linkages generated by the *Jaro* and *Jaccard* (cf. Fig. 9.5), precision and recall results of *Jaccard* were better than *Jaro*. Clearly, *Jaro* is less capable to identify the correct linkages between entities, and thus, it generates more linkages which are less certain. This generates more pairwise-linked entities which are now reflected in the size of factors.

756

757

758

759

760

761

762

763

764

765

766

*Time to Retrieve Possible Worlds.* Given that factors have different sizes, we measured the time needed to identify the possible world with the highest probability in respect to the factor size (Sect. 9.4.3). Figure 9.11 shows the required time for different factor sizes. As expected, for small factor sizes (i.e., 20–40 entity linkages) which is the dominating majority among the factors, the algorithm requires around 1 millisecond. For larger factor sizes, the algorithm requires more time, which however still remains below 4 ms.

767

768

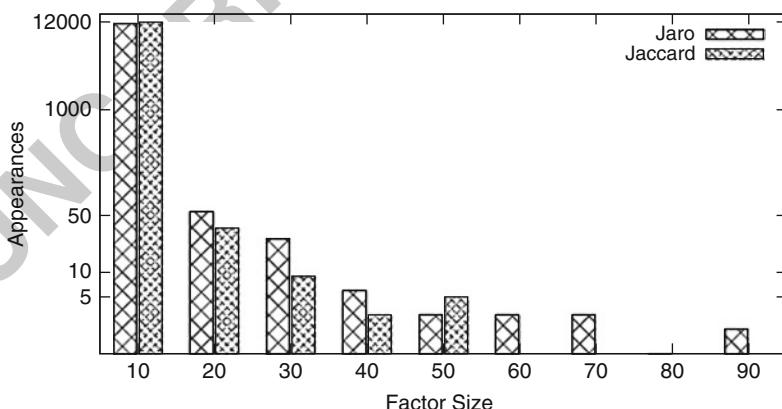
769

770

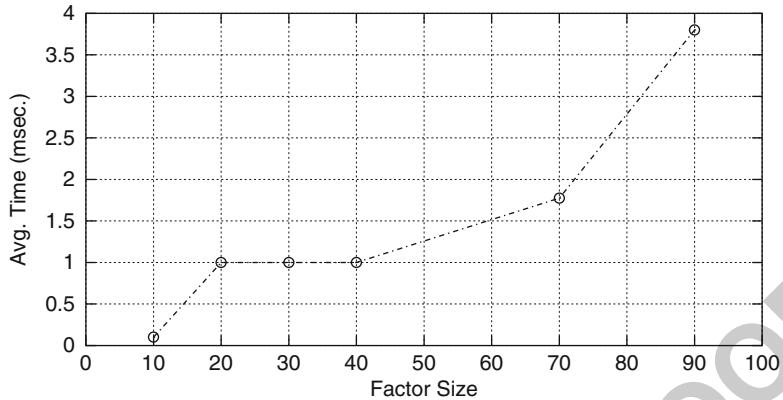
771

772

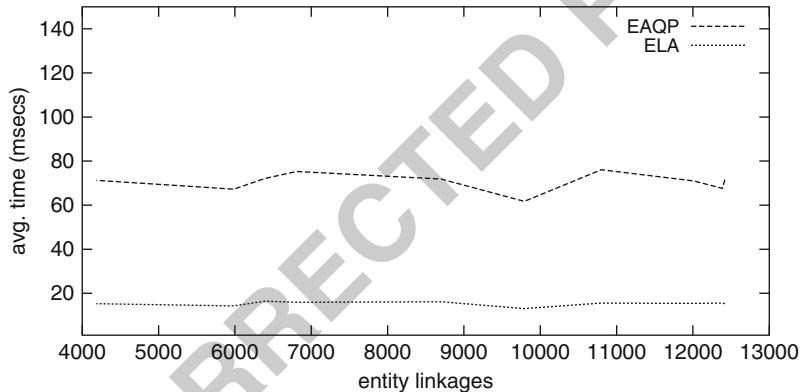
773



**Fig. 9.10** Appearance numbers for the factor sizes generated for our two movie datasets



**Fig. 9.11** Average time for computing the possible world with the maximum probability over factor sizes



**Fig. 9.12** Total time for processing queries over different numbers of entity linkages in dataset

**Execution Time.** Our final evaluation was to measure the time required for entity-aware query processing and also to compute the overhead that a system will have for offering this additional functionality. Figure 9.12 shows the average time taken to answer queries with and without our approach. We show time over different numbers of entity linkages in dataset. As expected, there is an increase in the time required with our approach, but this is relatively small and it remains under 70 ms. Furthermore, time does not increase as the dataset gets larger. On the contrary, query time remains stable even when the dataset is double the size. This behavior results from the effective grouping of linkages into factors which allows the algorithm to easily detect and use only a small subset of the linkages during query processing.

774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784

### 9.5.3 Evaluation Summary

785

Summarizing, the result of our experimental evaluation confirms the following:

786

- Incorporating entity-aware query processing in a system makes the system able to better handle the entity linkage problem and especially provide query answers which reflect the possible entity solutions for the current data. 787
- Our approach has a small overhead in time required for processing queries, but due to our efficient processing strategy, this cost remains low and constant even for large datasets with a large amount of entity linkages. 790

791

792

## 9.6 Conclusions

793

We have introduced a novel approach that allows on-the-fly entity-aware query processing in the presence of linkage information. Our approach can be applied on various data formats and structures using a generic entity representation. We explained how query processing can be performed efficiently over the entities and their possible linkages as these are generated by existing entity linkage techniques. Special focus was given on handling the uncertainty that appears in the entity linkage information as well as in the entity data. Our evaluation shows that the approach is both efficient and effective in answering entity queries. 801

794

795

796

797

798

799

800

801

We are currently investigating several directions to extend our approach. First, we would like to cover provenance information related to the possible linkage decisions and answers returned by querying. Also, we would like to investigate the implications of having conflicting information for the entity descriptions, as is typically the case for Web data, and to try out effective ways to deal with such conflicts. 802

802

803

804

805

806

## References

807

1. Adar, E., Re, C.: Managing uncertainty in social networks. IEEE Data Eng. Bull. 15–22 (2007) 808
2. Agrawal, P., Benjelloun, O., Sarma, A., Hayworth, C., Nabar, S., Sugihara, T., Widom, J.: Trio: a system for data, uncertainty, and lineage. VLDB, pp. 1151–1154 (2006) 810
3. Andritsos, P., Fuxman, A., Miller, R.: Clean answers over dirty databases: a probabilistic approach. ICDE (2006) 811
4. Antova, L., Koch, C., Olteanu, D.:  $10^{(10)^6}$  worlds and beyond: efficient representation and processing of incomplete information. VLDB J. **18**(5), 1021–1040 (2009) 813
5. Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S., Widomr, J., Jonas, J.: Swoosh: a generic approach to entity resolution. VLDB J. **18**(1), 255–276 (2009) 815
6. Bex, G., Neven, F., Vansumeren, S.: Inferring xml schema definitions from xml data. VLDB, pp. 998–1009 (2007) 817
7. Bhattacharya, I., Getoor, L.: Iterative record linkage for cleaning and integration. DMKD, pp. 11–18 (2004) 819

820

8. Bilenko, M., Mooney, R., Cohen, W., Ravikumar, P., Fienberg, S.: Adaptive name matching in information integration. *IEEE Intel. Syst.* **18**(5), 16–23 (2003) 821  
822
9. Cohen, W., Ravikumar, P., Fienberg, S.: A comparison of string distance metrics for name-matching tasks. *IIWeb*, pp. 73–78 (2003) 823  
824
10. Dalvi, N., Kumar, R., Pang, B., Ramakrishnan, R., Tomkins, A., Bohannon, P., Keerthi, S., Merugu, S.: A web of concepts. *PODS*, pp. 1–12 (2009) 825  
826
11. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. *VLDB J.* **16**(4), 523–544 (2007) 827  
828
12. Dalvi, N., Suciu, D.: Management of probabilistic data: foundations and challenges. *PODS*, pp. 1–12 (2007) 829  
830
13. Dasu, T., Johnson, T.: *Exploratory Data Mining and Data Cleaning*. Wiley, NY, USA (2003) 831
14. Doan, A., Halevy, A.Y.: Semantic integration research in the database community: a brief survey. *AI Mag.* **26**(1), 83–94 (2005) 832  
833
15. Doan, A., Lu, Y., Lee, Y., Han, J.: Object matching for information integration: a profiler-based approach. *IIWeb*, pp. 53–58 (2003) 834  
835
16. Domingos, P.: Multi-relational record linkage. Multi-relational data mining workshop co-located with KDD, pp. 31–48 (2004) 836  
837
17. Dong, X., Halevy, A., Madhavan, J.: Reference reconciliation in complex information spaces. SIGMOD conference, pp. 85–96 (2005) 838  
839
18. Dong, X., Halevy, A., Yu, C.: Data integration with uncertainty. *VLDB*, pp. 687–698 (2007) 840
19. Elmagarmid, A., Ipeirotis, P., Verykios, V.: Duplicate record detection: a survey. *IEEE Trans. Knowl. Data Eng.* **19**(1), 1–16 (2007) 841  
842
20. Getoor, L., Diehl, C.: Link mining: a survey. *SIGKDD explorations* (2005) 843
21. Gupta, R., Sarawagi, S.: Creating probabilistic databases from information extraction models. *VLDB*, pp. 965–976 (2006) 844  
845
22. Halevy, A., Franklin, M., Maier, D.: Principles of dataspace systems. *PODS*, pp. 1–9 (2006) 846
23. Hernández, M., Stolfo, S.: Real-world data is dirty: data cleansing and the merge/purge problem. *Data Mining Knowledge Dis.* **2**(1), 9–37 (1998) 847  
848
24. Ioannou, E., Nejdl, W., Niederée, C., Velegrakis, Y.: On-the-fly entity-aware query processing in the presence of linkage. *PVLDB* **3**(1), 429–438 (2010) 849  
850
25. Ioannou, E., Nejdl, W., Niederée, C., Velegrakis, Y.: LinkDB: a probabilistic linkage database system. SIGMOD conference, pp. 1307–1310 (2011) 851  
852
26. Ioannou, E., Niederée, C., Nejdl, W.: Probabilistic entity linkage for heterogeneous information spaces. *CAiSE*, pp. 302–316 (2008) 853  
854
27. Kalashnikov, D., Mehrotra, S.: Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Trans. Database Syst.* **31**(2), 716–767 (2006) 855  
856
28. Lenzerini, M.: Data integration: a theoretical perspective. *PODS*, pp. 233–246 (2002) 857
29. Morris, A., Velegrakis, Y., Bouquet, P.: Entity identification on the semantic web. *SWAP* (2008) 858
30. Papadakis, G., Ioannou, E., Niederée, C., Fankhauser, P.: Efficient entity resolution for large heterogeneous information spaces. *WSDM*, pp. 535–544 (2011) 859  
860
31. Rastogi, V., Dalvi, N., Garofalakis, M.: Large-scale collective entity matching. *PVLDB* **4**(4), 208–218 (2011) 861  
862
32. Re, C., Suciu, D.: Managing probabilistic data with MystiQ: the can-do, the could-do, and the can't-do. *SUM*, pp. 5–18 (2008) 863  
864
33. Sarawagi, S., Bhamidipaty, A.: Interactive deduplication using active learning. *KDD*, pp. 269–278 (2002) 865  
866
34. Sen, P., Deshpande, A.: Representing and querying correlated tuples in probabilistic databases. *ICDE*, pp. 596–605 (2007) 867  
868
35. Velegrakis, Y.: On the importance of updates in information integration and data exchange systems. *DBISP2P* (2008) 869  
870
36. Whang, S., Menestrina, D., Koutrika, G., Theobald, M., Garcia-Molina, H.: Entity resolution with iterative blocking. *SIGMOD Conference*, pp. 219–232 (2009) 871  
872

AUTHOR QUERIES

- AQ1. Kindly check the corresponding author.
- AQ2. Please check whether the changes made to this sentence retain the intended meaning.
- AQ3. Please confirm the insertion of He or in the sentence “She sends to the system...”

UNCORRECTED PROOF

# Chapter 10

## The Return of the Entity-Relationship Model: Ontological Query Answering

Andrea Calì, Georg Gottlob, and Andreas Pieris

### 10.1 Introduction

#### 10.1.1 Motivation

The Entity-Relationship (ER) model is a fundamental formalism for conceptual modeling in database design; it was introduced by Chen in his milestone paper [3], and it is now widely used, being flexible and easily understood by practitioners. With the rise of the semantic Web, conceptual modeling formalisms have gained importance again as *ontology formalisms*, in the semantic Web parlance. Ontologies and conceptual models are aimed at representing, rather than the structure of data, the domain of interest, that is, the fragment of the real world that is being represented by the data and the schema. A prominent formalism for modeling ontologies is *description logics (DLs)* [4], which are decidable fragments of first-order logic, particularly suitable for ontological modeling and querying. In particular, DL ontologies are sets of assertions describing sets of objects and (usually binary) relations among such sets, exactly in the same fashion as the ER model.

Recently, research on DLs has been focusing on the problem of *answering queries* under ontologies, that is, given a query  $q$ , an instance  $B$ , and an ontology  $\mathcal{K}$ , answering  $q$  under  $B$  and  $\mathcal{K}$  amounts to compute the answers that are *logically entailed* from  $B$  by using the assertions of  $\mathcal{K}$ . In this context, where data size

---

AQ1

A. Calì (✉)

Department of Computer Science and Information Systems, Birkbeck University of London,  
Malet St., London WC1E 7HX, UK

e-mail: [andrea@dcs.bbk.ac.uk](mailto:andrea@dcs.bbk.ac.uk)

G. Gottlob · A. Pieris

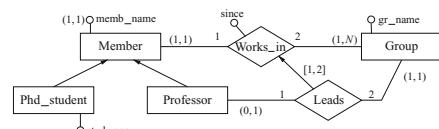
Department of Computer Science, University of Oxford, Wolfson Building, Parks Road,  
Oxford OX1 3QD, UK

e-mail: [georg.gottlob@cs.ox.ac.uk](mailto:georg.gottlob@cs.ox.ac.uk); [andreas.pieris@cs.ox.ac.uk](mailto:andreas.pieris@cs.ox.ac.uk)

is usually large, a central issue, the *data complexity* of query answering, i.e., the computational complexity with respect to the dataset  $B$  only, while the ontology  $\mathcal{K}$  and the query  $q$  are fixed.

For this reason, the tractable *DL-Lite* family [5, 6], for instance, has become a prominent family of DL languages. More precisely, DL-Lite languages are *first-order rewritable*, i.e., for every conjunctive query  $q$ , for every initial instance  $B$ , and for every DL-Lite knowledge base  $\mathcal{K}$ , it is possible to construct a first-order query  $q_{\mathcal{K}}$  (which, in particular, is a union of conjunctive queries), such that directly evaluating  $q_{\mathcal{K}}$  over  $B$  (without resorting to the knowledge base  $\mathcal{K}$  any more) returns the correct answers to  $q$  under  $B$  and  $\mathcal{K}$ . The first-order rewritability implies two desirable properties: (1) the data complexity of conjunctive query answering is very low, namely, in the highly tractable class  $AC_0$  (which is the complexity class of recognizing words in languages defined by constant-depth Boolean circuits with (unlimited fan-in) AND and OR gates), and (2) query answering can be performed efficiently by translating the rewritten query  $q_{\mathcal{K}}$  into SQL, thus taking advantage of the optimization capabilities of the underlying relational DBMS.

In this chapter, we consider the problem of conjunctive query answering under an extended version of the ER model, which we call  $ER^+$ . Our model is obtained from the original ER model by adding is-a relations among entities and relationships, functional and mandatory participation constraints on the participation of entities to relationships, and functional and mandatory constraints on attributes of entities and relationships. Figure 10.1 shows an  $ER^+$  schema, where the reader will recognize the familiar graphical notation of Chen's ER model. The schema describes members of a university department working in research groups. Ph.D. students and professors are both members. For instance, the constraint (1, 1) (min. 1, max. 1) on the participation of *Member* in *Works\_in* imposes that every instance of *Member* participates at least once (mandatory participation) and at most once (functional participation) in *Works\_in*. The constraint (0, 1) (min. 0, max. 1) on the participation of *Professor* in *Leads* imposes that every instance of *Professor* participates at most once (functional participation) in *Leads*. The constraint (1,  $N$ ) (min. 1, max. arbitrary) on the participation of *Group* in *Works\_in* imposes that every instance of *Group* participates at least once (mandatory participation) in *Works\_in*. The constraint (1, 1) on the attribute *memb\_name* states that such an attribute is mandatory (at least one value) and functional (at most one value). The is-a constraint among the relationships states that each professor works in the same group that (s)he leads, considering the components in the same order; components of relationships are ordered and marked by the integers 1 and 2. The notation [1, 2] indicates the permutation of the components involved in the containment.



**Fig. 10.1** An  $ER^+$  schema

In our setting, we admit arbitrary *permutations* of objects in an is-a between two relationships. For example, we can assert that each instance  $\langle a, b, c \rangle$  of a ternary relationship  $R_1$  is also an instance of another ternary relationship  $R_2$ , but the three objects (instances of the participating entities) appear in  $R_2$  in the order  $\langle c, a, b \rangle$ ; this would be indicated by the annotation  $[3, 1, 2]$  in the corresponding diagram. The addition of the permutation feature increases the complexity of query answering. The permutation is a common feature in DL-Lite and other semantic Web languages, where it is usually in the form of *inverse roles*, where roles are binary relations among *concepts* (sets). With this feature, we can express, for instance, the reflexive property of a relation; for example (in first-order logic),  $\forall X \forall Y \text{ brother}(X, Y) \rightarrow \text{brother}(Y, X)$ .

Similar to what is found in the literature on DL-Lite, we are interested in query answering under ontologies and, in particular, in the data complexity of (the decision version of) such problem in the presence of  $\text{ER}^+$  ontologies. We identify and study sublanguages of  $\text{ER}^+$  which are tractable in data complexity, and we also provide a complete complexity study (data and combined complexity) of query answering under such sublanguages.

### 10.1.2 Summary of the Chapter

We first introduce, in Sect. 10.3, the  $\text{ER}^+$  model and provide a formal semantics for it in terms of a relational representation; that is, each  $\text{ER}^+$  schema is encoded in a relational schema, plus a set of relational constraints—belonging to the well-known classes of *inclusion dependencies* and *key dependencies*. A set of inclusion and key dependencies in such a form that it can represent an  $\text{ER}^+$  schema is called a set of *conceptual dependencies* (CDs). We study different variants of the  $\text{ER}^+$  model in terms of the corresponding (sub)classes of CDs. Our central algorithmic tool is the *chase* [7, 8], widely adopted in the literature in the context of query containment and answering under relational constraints.

It turns out that  $\text{ER}^+$  is *not* first-order rewritable. The chief reason for this is the interaction between inclusion dependencies and key dependencies in a set of CDs, which mirrors the interaction between functional participation constraints and the other kinds of constraints in the corresponding  $\text{ER}^+$  schema. The central (semantic) notion here is then *separability*, which guarantees a decoupling between inclusion and key dependencies in a set of CDs: under a set  $\Sigma$  of separable CDs, answering a query  $q$  on an instance  $D$  is equivalent to answering  $q$  on  $D$  under the inclusion dependencies of  $\Sigma$  only. We provide a *syntactic* condition, in Sect. 10.4, which implies separability of CDs; we do this by defining *nonconflicting CDs* (and, consequently, nonconflicting  $\text{ER}^+$  schemata). We show that the aforementioned condition is not only sufficient but also necessary; this way, we do precisely characterize the class of separable  $\text{ER}^+$  schemata.

We provide, in Sect. 10.5, a complexity analysis of the query answering problem for the following three cases: (1) nonconflicting  $\text{ER}^+$  schemata; (2) nonconflicting

**Table 10.1** Summary of complexity results

Language	Arity	Perm.	Combined	Data	
Nonconflicting $\text{ER}^+$	Any	Yes	PSPACE-complete	$\text{AC}_0$	t46.1
			UB: [8]	UB: [9]	t46.2
Nonconflicting $\text{ER}_i^+$	Any	No	NP-complete	$\text{AC}_0$	t46.3
			LB: [10]	UB: [9]	t46.4
Nonconflicting $\text{ER}_B^+$	Bounded	Yes	NP-complete	$\text{AC}_0$	t46.5
			UB: [8] LB: [10]	UB: [9]	t46.6
					t46.7

$\text{ER}^+$  schemata with *identity permutations*, i.e., schemata where every is-a among relationships has the identity permutation associated with it (nonconflicting  $\text{ER}_i^+$ ); and (3) nonconflicting  $\text{ER}^+$  schemata with the arity of relationships bounded by a constant (nonconflicting  $\text{ER}_B^+$ ). Observe that  $\text{ER}^+$  properly generalizes both  $\text{ER}_i^+$  and  $\text{ER}_B^+$ . The complexity results shown in this chapter are summarized in Table 10.1. When results in the table are already known from the literature, this is indicated (UB and LB stand for upper and lower bound, respectively). Notice that *all* nonconflicting variants of  $\text{ER}^+$  enjoy first-order rewritability, which in turn guarantees  $\text{AC}_0$  data complexity.

We then turn our attention, in Sect. 10.6, to *negative constraints*, which allow us to express, for instance, nonparticipation of an entity to a relationship, but also more general assertions. We show that the addition of negative constraints does not increase the complexity of query answering.

Last but not least, in Sect. 10.7, we mention that the tractable  $\text{ER}^+$  languages we present here have most DL-Lite languages as special cases. This confirms the usefulness of  $\text{ER}^+$  in ontology modeling and ontology-based query answering.

### 10.1.3 Related Work

Our model is based on Chen's ER model [3]. The extended model considered in [16] includes the generalization and full aggregation constructs, while the one in [15] allows for arbitrary data types, optional and multivalued attributes, a concept for specialization and generalization, and complex structured entity types. Markowitz and Makowsky [17] considers relational schemata with referential integrity constraints that can be associated with an extended ER schema.

The paper [18] gives a formal first-order semantics to the ER model. Reasoning tasks under theories expressed in different variants of the ER model, whose constructs are given a formal semantics, are studied in [19].

Query answering under ER ontologies is tackled in [20], where the proposed ER language is strictly less expressive than the ones presented in this chapter; the problem is studied in the context of data integration systems, and a first-order rewriting technique is proposed, later extended in [9]. The notion of separability was first introduced in [11]. Sufficient conditions for separability appear, for instance, in [26, 27].

The formalism of [21], which is also studied with respect to query answering, 134  
 is more expressive than  $\text{ER}^+$ , and it does not offer the same tractability of query 135  
 answering. 136

Other works [8, 22] consider query containment, which is tightly related to the 137  
 query answering problem in the case of incomplete information. Calvanese et al. 138  
 [22] considers query containment in a formalism similar to  $\text{ER}^+$ , but with less 139  
 expressive negative constraints, and eventually incomparable to  $\text{ER}^+$ ; the combined 140  
 complexity is higher than the one under nonconflicting  $\text{ER}^+$ , and data complexity 141  
 is not studied. 142

Our work is tightly related to the semantic Web and, in general, to Web data 143  
 management. The World Wide Web Consortium (W3C) defines several standards, 144  
 including the *resource description framework (RDF)* for the data layer; the *Web 145  
 ontology language (OWL)*, based on description logics (DLs), for the ontology 146  
 layer; and the *rule interchange format (RIF)*, currently being defined as a standard, 147  
 for the rule layer. Several ontology languages have been proposed, many of 148  
 which are more expressive than  $\text{ER}^+$ , while being less efficient in terms of 149  
 query answering. The  $\text{ER}^+$  model is intended to be a viable alternative to the 150  
 most prominent *tractable* OWL sublanguages. Our  $\text{ER}^+$  variants offer the same, 151  
 desirable tractability properties (first-order rewritability, in particular), while in 152  
 addition avoiding the description logic syntax, which could turn out to be slightly 153  
 awkward to database practitioners. Due to the popularity of the ER model in the 154  
 industry, and to its intuitive graphical representation, the  $\text{ER}^+$  formalism is likely 155  
 to be adopted as ontology language, especially when employed in visual design 156  
 tools. More specifically, the languages of the DL-Lite family [5, 6], as already 157  
 explained, enjoy the desirable property of tractable conjunctive query answering, in 158  
 particular in  $\text{AC}_0$  in data complexity, again ensured by their first-order rewritability. 159  
 Both our formalism nonconflicting  $\text{ER}^+$  and nonconflicting  $\text{ER}_B^+$ , with the addition 160  
 of negative constraints, which do not increase the complexity of query answering, 161  
 properly generalize the languages  $\text{DL-Lite}_{\mathcal{F}}$  and  $\text{DL-Lite}_{\mathcal{R}}$ ; at the same time, the 162  
 data complexity of query answering is the same as that of the DL-Lite languages. 163

The notion of *chase* [7, 8, 11] of a database against a set of inclusion dependencies 164  
 is crucial in ontological query answering; such notion has been extended to 165  
 expressive rules called *tuple-generating dependencies (TGDs)* [23, 24]. In particular, 166  
 Fagin et al. [23] considers the *data exchange* problem, where a target database is 167  
 materialized starting from a source database and a mapping expressed as a set of 168  
 TGDs; in this setting, the chase procedure needs to terminate [24, 25]. However, this 169  
 is not appropriate for ontological databases. The first work to tackle the problem of 170  
 a nonterminating chase was [8]. 171

Recent works [14, 26–28] deal with so-called *tuple-generating dependencies*, 172  
 rules that constitute the *Datalog $^\pm$*  family of languages, some of which are first-order 173  
 rewritable. Datalog $^\pm$  languages (enriched with nonconflicting key dependencies as 174  
 in [2] or [26]) are not capable of capturing nonconflicting  $\text{ER}^+$  schemata. For an 175  
 overview of some Datalog $^\pm$  languages, we refer the reader to [29]. 176

Finally, the work [30] deals with general (nonseparable)  $\text{ER}^+$  schemata: a 177  
 technique for query rewriting into recursive Datalog is presented, and complexity 178  
 bounds are significantly higher than those shown here. 179

## 10.2 Preliminaries

180

In this section, we recall some basics on databases, queries, dependencies, and the chase procedure. 181  
182

### 10.2.1 General

183

We define the following pairwise disjoint (infinite) sets of symbols: (1) a set  $\Gamma$  of *constants* (constitute the “normal” domain of a database), (2) a set  $\Gamma_N$  of *labeled nulls* (used as placeholders for unknown values, and thus can be also seen as variables), and (3) a set  $\Gamma_V$  of *variables* (used in queries and dependencies). 184  
185  
186  
187  
Different constants represent different objects (*unique name assumption*), while 188  
different nulls may represent the same object. A lexicographic order is defined 189  
on  $\Gamma \cup \Gamma_N$ , such that every value in  $\Gamma_N$  follows all those in  $\Gamma$ . We denote by 190  
 $\mathbf{X}$  sequences (or sets) of variables and constants  $X_1, \dots, X_k$ , where  $k > 0$ ; it will 191  
be clear from the context when we regard  $\mathbf{X}$  as a sequence of symbols, and when as 192  
a set of symbols. For an integer  $n > 1$ , we denote by  $[n]$  the set  $\{1, \dots, n\}$ . 193

A relational schema  $\mathcal{R}$  (or simply *schema*) is a set of relational symbols (or 194  
predicates), each with its associated arity. We write  $r/n$  to denote that the predicate 195  
 $r$  has arity  $n$  and we refer to the integer  $n$  by *arity*( $r$ ). A position  $r[i]$ , in a schema 196  
 $\mathcal{R}$ , is identified by a predicate  $r \in \mathcal{R}$  and its  $i$ th argument (or attribute). A term  $t$  is 197  
either a constant or a null or a variable. An atomic formula (or simply *atom*) has the 198  
form  $r(t_1, \dots, t_n)$ , where  $r/n$  is a relation, and  $t_1, \dots, t_n$  are terms. For an atom  $a$ , 199  
we denote as *dom*( $a$ ) and *pred*( $a$ ) the set of its terms and its predicate, respectively. 200  
These notations naturally extend to sets and conjunctions of atoms. Conjunctions of 201  
atoms will be often identified with the sets of their atoms. 202

A substitution from a set  $S_1 \subset \Gamma \cup \Gamma_N \cup \Gamma_V$  to a set  $S_2 \subset \Gamma \cup \Gamma_N \cup \Gamma_V$  is a function 203  
 $h : S_1 \rightarrow S_2$  defined as follows: (1)  $\emptyset$  is a substitution, and (2) if  $h$  is a substitution, 204  
then  $h \cup \{X \rightarrow Y\}$  is a substitution, where  $X \in S_1$  and  $Y \in S_2$ . A homomorphism 205  
from a set of atoms  $A_1$  to a set of atoms  $A_2$  is a substitution  $h : \text{dom}(A_1) \rightarrow$  206  
 $\text{dom}(A_2)$  such that: (1) if  $t \in \Gamma$ , then  $h(t) = t$ , and (2) if  $r(t_1, \dots, t_n) \in A_1$ , 207  
then  $h(r(t_1, \dots, t_n)) = r(h(t_1), \dots, h(t_n)) \in A_2$ . If there exists a homomorphism 208  
from  $A_1$  to  $A_2$  and vice versa, then  $A_1$  and  $A_2$  are *homomorphically equivalent*. The 209  
notion of homomorphism naturally extends to conjunctions of atoms. 210

### 10.2.2 Databases and Queries

211

A relational instance (or simply *instance*)  $I$  for a schema  $\mathcal{R}$  is a (possibly infinite) 212  
set of atoms of the form  $r(\mathbf{t})$ , where  $r/n \in \mathcal{R}$  and  $\mathbf{t} \in (\Gamma \cup \Gamma_N)^n$ . We denote 213  
as  $r(I)$  the set  $\{\mathbf{t} \mid r(\mathbf{t}) \in I\}$ . A database  $D$  for  $\mathcal{R}$  is a finite instance for  $\mathcal{R}$  such 214

that  $\text{dom}(D) \subset \Gamma$ . For simplicity, we assume source databases which contain only ground atoms, i.e., atoms whose arguments are constants of  $\Gamma$ . However, it is not difficult to show that the results of this chapter hold even if we allow incomplete source databases which may include labeled nulls.

A *conjunctive query (CQ)*  $q$  of arity  $n$  over a schema  $\mathcal{R}$ , written as  $q/n$ , is an assertion of the form  $p(\mathbf{X}) \leftarrow \varphi(\mathbf{X}, \mathbf{Y})$ , where  $\varphi(\mathbf{X}, \mathbf{Y})$  is a conjunction of atoms over  $\mathcal{R}$ , and  $p$  is an  $n$ -ary predicate not occurring in  $\mathcal{R}$ .  $\varphi(\mathbf{X}, \mathbf{Y})$  is called the *body* of  $q$ , denoted as  $\text{body}(q)$ . The *answer* to a CQ  $q/n$  of the above form over an instance  $I$ , denoted as  $q(I)$ , is the set of all  $n$ -tuples  $\mathbf{t} \in \Gamma^n$  for which there exists a homomorphism  $h : \mathbf{X} \cup \mathbf{Y} \rightarrow \Gamma \cup \Gamma_N$  such that  $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$  and  $h(\mathbf{X}) = \mathbf{t}$ .

### 10.2.3 Dependencies

225

A *tuple-generating dependency (TGD)*  $\sigma$  over  $\mathcal{R}$  is a first-order formula of the form  $\forall \mathbf{X} \forall \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \psi(\mathbf{X}, \mathbf{Z})$ , where  $\varphi(\mathbf{X}, \mathbf{Y})$  and  $\psi(\mathbf{X}, \mathbf{Z})$  are conjunctions of atoms over  $\mathcal{R}$ , called the *body* and the *head* of  $\sigma$ , denoted as  $\text{body}(\sigma)$  and  $\text{head}(\sigma)$ , respectively. Henceforth, for notational convenience, we will omit the universal quantifiers in front of TGDs. Such  $\sigma$  is satisfied by an instance  $I$  for  $\mathcal{R}$  if whenever there exists a homomorphism  $h$  such that  $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$ , then there exists an extension  $h'$  of  $h$ , i.e.,  $h' \supseteq h$ , such that  $h'(\psi(\mathbf{X}, \mathbf{Z})) \subseteq I$ .

A *key dependency (KD)*  $\kappa$  over  $\mathcal{R}$  is an assertion of the form  $\text{key}(r) = \mathbf{A}$ , where  $r \in \mathcal{R}$  and  $\mathbf{A}$  is a set of attributes of  $r$ . Such  $\kappa$  is satisfied by an instance  $I$  for  $\mathcal{R}$  if for each pair of distinct tuples  $\mathbf{t}_1, \mathbf{t}_2 \in r(I)$ ,  $\mathbf{t}_1[\mathbf{A}] \neq \mathbf{t}_2[\mathbf{A}]$ , where  $\mathbf{t}[\mathbf{A}]$  is the projection of tuple  $\mathbf{t}$  over  $\mathbf{A}$ .

A *negative constraint (NC)*  $v$  over  $\mathcal{R}$  is a first-order formula  $\forall \mathbf{X} \varphi(\mathbf{X}) \rightarrow \perp$ , where  $\varphi(\mathbf{X})$  is a conjunction of atoms over  $\mathcal{R}$ , called the *body* of  $v$  and denoted as  $\text{body}(v)$ , and  $\perp$  is the truth constant *false*. For brevity, we will omit the universal quantifiers in front of NCs and assume such a quantification implicitly. Such  $v$  is satisfied by an instance  $I$  if there is no homomorphism  $h$  such that  $h(\varphi(\mathbf{X})) \subseteq I$ .

Given an instance  $I$  and a dependency  $\delta$ , we write  $I \models \delta$  (resp.,  $I \not\models \delta$ ) if  $I$  satisfies (resp., violates)  $\delta$ . Given a set of dependencies  $\Sigma$ , we say that  $I$  satisfies  $\Sigma$ , written as  $I \models \Sigma$ , if for each  $\delta \in \Sigma$  it holds that  $I \models \delta$ . Conversely, we say that  $I$  violates  $\Sigma$ , written as  $I \not\models \Sigma$ , if there exists  $\delta \in \Sigma$  such that  $I \not\models \delta$ .

### 10.2.4 Query Answering Under Dependencies

246

Let us now introduce the problem of query answering under dependencies. Given a database  $D$  and a set of dependencies  $\Sigma$ , the answers we consider are those that are true in *all* models of  $D$  with respect to  $\Sigma$ , i.e., all instances that contain  $D$  and satisfy the dependencies of  $\Sigma$ . Formally, the *models* of  $D$  with respect to  $\Sigma$ , denoted as  $\text{mods}(D, \Sigma)$ , are the set of all instances  $I$  such that  $I \models D \cup \Sigma$ . The *answer* to

a CQ  $q/n$  with respect to  $D$  and  $\Sigma$  is the set of  $n$ -tuples  $ans(q, D, \Sigma) = \{\mathbf{t} \mid \mathbf{t} \in q(I), \text{ for each } I \in mods(D, \Sigma)\}$ . 252  
253

The *decision problem* associated with conjunctive query answering under dependencies, dubbed *CQ-Ans*, is defined as follows: given a CQ  $q/n$  over a schema  $\mathcal{R}$ , a database  $D$  for  $\mathcal{R}$ , a set  $\Sigma$  of dependencies over  $\mathcal{R}$ , and an  $n$ -tuple  $\mathbf{t} \in \Gamma^n$ , decide whether  $\mathbf{t} \in ans(q, D, \Sigma)$ . Following Vardi's taxonomy [31], the *data complexity* of the above problem is the complexity calculated taking only the database as input, while the query and the set of dependencies are considered fixed. The *combined complexity* is the complexity calculated considering as input, together with the database, also the query and the set of dependencies. 254  
255  
256  
257  
258  
259  
260  
261

For the moment, we put NCs aside and deal only with TGDs and KDs; we shall return to consider also NCs in Sect. 10.6. 262  
263

### 10.2.5 The Chase Procedure

264

The *chase procedure* (or simply *chase*) is a fundamental algorithmic tool introduced for checking implication of dependencies [7], and later for checking query containment [8]. The chase is a process of repairing a database with respect to a set of dependencies so that the resulting instance satisfies the dependencies. We shall use the term *chase* interchangeably for both the procedure and its result. The chase works on an instance through the so-called TGD and KD *chase rules*. 265  
266  
267  
268  
269  
270

**TGD Chase Rule.** Consider an instance  $I$  for a schema  $\mathcal{R}$ , and a TGD  $\sigma = \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \psi(\mathbf{X}, \mathbf{Z})$  over  $\mathcal{R}$ . If  $\sigma$  is *applicable* to  $I$ , i.e., there exists a homomorphism  $h$  such that  $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$ , then (1) define  $h' \supseteq h$  such that  $h'(Z_i) = z_i$  for each  $Z_i \in \mathbf{Z}$ , where  $z_i \in \Gamma_N$  is a “fresh” labeled null not introduced before, and following lexicographically all those introduced so far, and (2) add to  $I$  the set of atoms in  $h'(\psi(\mathbf{X}, \mathbf{Z}))$ , if not already in  $I$ . 271  
272  
273  
274  
275  
276

**KD Chase Rule.** Consider an instance  $I$  for a schema  $\mathcal{R}$ , and a KD  $\kappa$  of the form  $key(r) = \mathbf{A}$  over  $\mathcal{R}$ . If  $\kappa$  is *applicable* to  $I$ , i.e., there are two (distinct) tuples  $\mathbf{t}_1, \mathbf{t}_2 \in r(I)$  such that  $\mathbf{t}_1[\mathbf{A}] = \mathbf{t}_2[\mathbf{A}]$ , then (1) if there exists  $i \notin \mathbf{A}$  such that both  $\mathbf{t}_1[i]$  and  $\mathbf{t}_2[i]$  are constants of  $\Gamma$ , then there is a *hard violation* of  $\kappa$  and the chase *fails*, otherwise; (2) for each  $i \notin \mathbf{A}$ , either replace each occurrence of  $\mathbf{t}_1[i]$  with  $\mathbf{t}_2[i]$ , if the former follows lexicographically the latter, or vice versa. 277  
278  
279  
280  
281  
282

Given a database  $D$  for a schema  $\mathcal{R}$  and set  $\Sigma = \Sigma_T \cup \Sigma_K$  over  $\mathcal{R}$ , where  $\Sigma_T$  are TGDs and  $\Sigma_K$  are KDs, the chase algorithm for  $D$  with respect to  $\Sigma$  consists of an exhaustive application of the chase rules, which leads to an (possibly infinite) instance, denoted as  $chase(D, \Sigma)$ . In particular,  $chase(D, \Sigma)$  is the instance constructed by applying (1) the TGD chase rule once, and (2) the KD chase rule as long as it is applicable (i.e., until a fixpoint is reached). We assume that the chase algorithm is *fair*, i.e., each TGD that must be applied during the construction of  $chase(D, \Sigma)$  is eventually applied. We denote as  $chase^{[k]}(D, \Sigma)$  the *initial segment* 283  
284  
285  
286  
287  
288  
289  
290

AQ2

of the chase of  $D$  with respect to  $\Sigma$  obtained by applying  $k$  times the (TGD or KD) chase rule during the construction of the chase. 291  
292

*Example 10.1.* Let  $\mathcal{R} = \{r, s\}$ . Consider the set  $\Sigma$  of TGDs and KDs over  $\mathcal{R}$  293 constituted by the TGDs  $\sigma_1 = r(X, Y) \rightarrow \exists Z r(Z, X), s(Z)$  and  $\sigma_2 = r(X, Y) \rightarrow$  294  $r(Y, X)$ , and the KD  $\kappa$  of the form  $\text{key}(r) = \{2\}$ . Let  $D = \{r(a, b)\}$ . During the 295 construction of  $\text{chase}(D, \Sigma)$ , we first apply  $\sigma_1$ , and we add the atoms  $r(z_1, a), s(z_1)$ , 296 where  $z_1$  is a “fresh” null of  $\Gamma_N$ . Moreover,  $\sigma_2$  is applicable and we add the atom 297  $r(b, a)$ . Now, the KD  $\kappa$  is applicable and we replace each occurrence of  $z_1$  with the 298 constant  $b$ ; thus, we get the atom  $s(b)$ . ■ 299

The fairness assumption allows us to show that the chase of  $D$  with respect to 300  $\Sigma$  is a *universal model* of  $D$  with respect to  $\Sigma$ , i.e., for each  $I \in \text{mods}(D, \Sigma)$ , 301 there exists a homomorphism from  $\text{chase}(D, \Sigma)$  to  $I$  (see, e.g., [23, 24]). Thus, 302 the answer to a CQ  $q$  with respect to  $D$  and  $\Sigma$ , if the chase does not fail, can be 303 obtained by evaluating  $q$  over  $\text{chase}(D, \Sigma)$ , and discarding tuples containing at least 304 one null [23]. If the chase fails, then  $\text{mods}(D, \Sigma) = \emptyset$  and  $\text{ans}(q, D, \Sigma) = \Gamma^n$ . 305

**Theorem 10.1.** Consider a CQ  $q/n$  over a schema  $\mathcal{R}$ , a database  $D$  for  $\mathcal{R}$ , a set 306  $\Sigma$  of TGDs and KDs over  $\mathcal{R}$ , and an  $n$ -tuple  $\mathbf{t} \in \Gamma^n$ . It holds that  $\mathbf{t} \in \text{ans}(q, D, \Sigma)$  307 if and only if  $\mathbf{t} \in q(\text{chase}(D, \Sigma))$  or  $\text{chase}(D, \Sigma)$  fails. 308

## 10.3 The Conceptual Model $\mathbf{ER}^+$

309

In this section, we present the conceptual model we shall deal with in this 310 paper, and we give its semantics in terms of relational schemata with dependen- 311 cies. This model, which is called  $\mathbf{ER}^+$ , incorporates the basic features of the 312 Entity-Relationship model [3] and OO models, including subset (or is-a) constraints 313 on both entities and relationships. It is an extension of the one presented in [20], and 314 we use a notation analogous to that of [20]. 315

### 10.3.1 Syntax of $\mathbf{ER}^+$

316

The schemata expressed in the  $\mathbf{ER}^+$  model are called  $\mathbf{ER}^+$  *schemata*. An 317  $\mathbf{ER}^+$  schema consists of a collection of entity, relationship, and attribute definitions 318 over an alphabet of symbols partitioned into a set of entity symbols  $\text{Ent}$ , a set of 319 relationship symbols  $\text{Rel}$ , and a set of attribute symbols  $\text{Att}$ . 320

An *entity definition* has the form 321

entity $E$	322
isa: $E_1, \dots, E_k$	323
participates( $\geq 1$ ): $R_1 : c_1, \dots, R_\ell : c_\ell$	324
participates( $\leq 1$ ): $R'_1 : c'_1, \dots, R'_m : c'_m$ ,	325

where (1)  $E \in Ent$  is the entity to be defined, (2) the **isa** clause specifies the set 326  
 of entities to which  $E$  is related via is-a, i.e., the set of entities that are supersets 327  
 of  $E$ , (3) the **participates( $\geq 1$ )** clause specifies that an instance of the entity  $E$  328  
 must necessarily participate in relationship  $R_i \in Rel$  as the  $c_i$ -th component, and 329  
 (4) the **participates( $\leq 1$ )** clause specifies that an instance of the entity  $E$  cannot 330  
 participate in relationship  $R'_i \in Rel$  as the  $c'_i$ -th component more than once. The 331  
**isa**, **participates( $\geq 1$ )** and **participates( $\leq 1$ )** clauses are optional. 332

A *relationship definition* has the form 333

**relationship**  $R$  **among**  $E_1, \dots, E_n$  334  
**isa:**  $R_1[j_{11}, \dots, j_{1n}], \dots, R_\ell[j_{\ell 1}, \dots, j_{\ell n}]$ , 335

where (1)  $R \in Rel$  is the relationship to be defined; (2) the entities of  $Ent$  listed 336  
 in the **among** clause are those among which the relationship  $R$  is defined ( $n$  is the 337  
*arity* of  $R$ ), i.e., the  $i$ th component of  $R$  is an instance of the entity  $E_i$ ; and (3) 338  
 the **isa** clause specifies the set of relationships to which  $R$  is related via is-a, i.e., 339  
 the set of relationships that are supersets of  $R$ ; for each relationship  $R_i$ , we specify 340  
 in square brackets how the components of  $R$  are related to those of  $R_i$ , by giving 341  
 a permutation  $[j_{i1}, \dots, j_{in}]$  of the set  $\{1, \dots, n\}$ . Henceforth, for brevity, given an 342  
 integer  $n > 0$  we will denote by  $[n]$  the set  $\{1, \dots, n\}$ . The **isa** clause is optional. 343

An *attribute definition* has the form 344

**attribute**  $A$  **of**  $X$  345  
**qualification,** 346

where (1)  $A \in Att$  is the attribute to be defined, (2)  $X \in (Ent \cup Rel)$  is either the entity 347  
 or the relationship to which the attribute is associated, and (3) *qualification* consists 348  
 of none, one, or both of the keywords **mandatory** and **functional**, specifying, 349  
 respectively, that each instance of  $X$  needs to have at least one value for attribute 350  
 $A$ , and that each instance of  $X$  has a unique value for  $A$ . If both **mandatory** 351  
 and **functional** are missing, the attribute is assumed by default to be optional and 352  
 multivalued, respectively. 353

We assume, without loss of generality, that each attribute is associated with a 354  
 unique entity or relationship, i.e., different entities and relationships have disjoint 355  
 sets of attributes. Also, for simplicity, we assume that attributes have atomic values. 356

*Example 10.2.* Consider the  $ER^+$  schema  $\mathcal{C}$  defined as follows: 357

<b>entity</b> <i>Member</i>	358
<b>participates(<math>\geq 1</math>)</b> : <i>Works_in</i> : 1	359
<b>participates(<math>\leq 1</math>)</b> : <i>Works_in</i> : 1	360
<b>entity</b> <i>Phd_student</i>	361
<b>isa</b> : <i>Member</i>	362
<b>entity</b> <i>Professor</i>	363
<b>isa</b> : <i>Member</i>	364
<b>participates(<math>\leq 1</math>)</b> : <i>Leads</i> : 1	365
<b>entity</b> <i>Group</i>	366
<b>participates(<math>\geq 1</math>)</b> : <i>Works_in</i> : 2, <i>Leads</i> : 2	367

participates( $\leq 1$ ): <i>Leads</i> : 2	368
relationship <i>Works_in</i> among <i>Member</i> , <i>Group</i>	369
relationship <i>Leads</i> among <i>Professor</i> , <i>Group</i>	370
isa: <i>Works_in</i> [1,2]	371
attribute <i>memb_name</i> of <i>Member</i>	372
mandatory, functional	373
attribute <i>stud_gpa</i> of <i>Phd_student</i>	374
attribute <i>gr_name</i> of <i>Group</i>	375
attribute <i>since</i> of <i>Works_in</i> .	376
■ 377	

It is easy to verify that  $\mathcal{C}$  is actually the schema depicted in Fig. 10.1.

### 10.3.2 Semantics of $\text{ER}^+$

378

The semantics of an  $\text{ER}^+$  schema  $\mathcal{C}$  is defined by associating a relational schema  $\mathcal{R}$  with it, and then specifying when a database for  $\mathcal{R}$  satisfies all the constraints imposed by the constructs of  $\mathcal{C}$ . We first define the relational schema that represents the so-called *concepts*, i.e., entities, relationships, and attributes, of an  $\text{ER}^+$  schema  $\mathcal{C}$  as follows: (1) each entity  $E$  in  $\mathcal{C}$  has an associated predicate  $e/1$ ; intuitively,  $e(c)$  asserts that  $c$  is an instance of entity  $E$ ; (2) each attribute  $A$  of an entity  $E$  in  $\mathcal{C}$  has an associated predicate  $a/2$ ; intuitively,  $a(c, d)$  asserts that  $d$  is the value of attribute  $A$  (of some entity  $E$ ) associated with  $c$ , where  $c$  is an instance of  $E$ ; (3) each relationship  $R$  of arity  $n$  in  $\mathcal{C}$  has an associated predicate  $r/n$ ; intuitively,  $r(c_1, \dots, c_n)$  asserts that  $\langle c_1, \dots, c_n \rangle$  is an instance of relationship  $R$  (among entities  $E_1, \dots, E_n$ ), where  $c_1, \dots, c_n$  are instances of  $E_1, \dots, E_n$ , respectively; and (4) each attribute  $A$  of a relationship  $R$  of arity  $n$  in  $\mathcal{C}$  has an associated predicate  $a/(n+1)$ ; intuitively,  $a(c_1, \dots, c_n, d)$  asserts that  $d$  is the value of attribute  $A$  (of some relationship  $R$  of arity  $n$ ) associated with the instance  $\langle c_1, \dots, c_n \rangle$  of  $R$ . ■ 392

*Example 10.3.* Consider the  $\text{ER}^+$  schema  $\mathcal{C}$  given in Example 10.2. The relational schema  $\mathcal{R}$  associated with  $\mathcal{C}$  consists of the predicates *member*/1, *phd\_student*/1, *professor*/1, *group*/1, *works\_in*/2, *leads*/2, *memb\_name*/2, *stud\_gpa*/2, *gr\_name*/2, *since*/3. Note that queries over an  $\text{ER}^+$  schema are queries over the relational schema associated with it. The CQ  $p(B) \leftarrow \text{phd\_student}(A), \text{memb\_name}(A, B)$ ,  $\text{works\_in}(A, C)$ ,  $\text{since}(A, C, 2006)$ ,  $\text{gr\_name}(C, db)$  asks for the names of the students who work in the “db” group since 2006. ■ 399

Once we have defined the relational schema  $\mathcal{R}$  for an  $\text{ER}^+$  schema  $\mathcal{C}$ , we give the semantics of each construct of  $\mathcal{C}$ . We do that by using the dependencies introduced in Sect. 10.2.3, as shown in Table 10.2 (the relationships have arity  $n$ ). ■ 402

**Definition 10.1.** Consider an  $\text{ER}^+$  schema  $\mathcal{C}$ , and let  $\mathcal{R}$  be the relational schema associated with it. The set of TGDs and KDs over  $\mathcal{R}$  obtained from  $\mathcal{C}$  as described above is called the set of *conceptual dependencies* (*CDs*) associated with  $\mathcal{C}$ . ■ 405

**Table 10.2** Derivation of relational constraints from an  $\text{ER}^+$  schema

$\text{ER}^+$ construct	Relational constraint	
Attribute $A$ for an entity $E$	$a(X, Y) \rightarrow e(X)$	t47.1
Attribute $A$ for a relationship $R$	$a(X_1, \dots, X_n, Y) \rightarrow r(X_1, \dots, X_n)$	t47.2
Rel. $R$ with entity $E$ as $i$ th component	$r(X_1, \dots, X_n) \rightarrow e(X_i)$	t47.3
Mandatory attribute $A$ of entity $E$	$e(X) \rightarrow \exists Y a(X, Y)$	t47.4
Mandatory attribute $A$ of relationship $R$	$r(X_1, \dots, X_n) \rightarrow \exists Y a(X_1, \dots, X_n, Y)$	t47.5
Functional attribute $A$ of an entity	$key(a) = \{1\}$ ( $a$ has arity 2)	t47.6
Functional attribute $A$ of a relationship	$key(a) = \{1, \dots, n\}$ ( $a$ has arity $n + 1$ )	t47.7
Is-a between entities $E_1$ and $E_2$	$e_1(X) \rightarrow e_2(X)$	t47.8
Is-a between relationships $R_1$ and $R_2$	$r_1(X_1, \dots, X_n) \rightarrow r_2(X_{i_1}, \dots, X_{i_n})$	t47.9
where components $1, \dots, n$ of $R_1$ correspond to components $i_1, \dots, i_n$ of $R_2$		t47.10
Mandatory part. of $E$ in $R$ ( $i$ th comp.)	$e(X_i) \rightarrow \exists X r(X_1, \dots, X_n)$	t47.11
Functional part. of $E$ in $R$ ( $i$ th comp.)	$key(r) = \{i\}$	t47.12
		t47.13
		t47.14

*Example 10.4.* Consider the  $\text{ER}^+$  schema  $\mathcal{C}$  given in Example 10.2. The relational schema  $\mathcal{R}$  associated with  $\mathcal{C}$  is given in Example 10.3. The set  $\Sigma$  of CDs over  $\mathcal{R}$  associated with  $\mathcal{C}$  is the following:

$$\begin{aligned}
 & member(X) \rightarrow \exists Y works\_in(X, Y) & leads(X, Y) \rightarrow professor(X) \\
 & key(works\_in) = \{1\} & leads(X, Y) \rightarrow group(Y) \\
 & phd\_student(X) \rightarrow member(X) & leads(X, Y) \rightarrow works\_in(X, Y) \\
 & professor(X) \rightarrow member(X) & memb\_name(X, Y) \rightarrow member(X) \\
 & key(leads) = \{1\} & member(X) \rightarrow \exists Y memb\_name(X, Y) \\
 & group(X) \rightarrow \exists Y works\_in(Y, X) & key(memb\_name) = \{1\}, \\
 & group(X) \rightarrow \exists Y leads(Y, X) & stud\_gpa(X, Y) \rightarrow phd\_student(X) \\
 & key(leads) = \{2\} & gr\_name(X, Y) \rightarrow group(X) \\
 & works\_in(X, Y) \rightarrow member(X), & since(X, Y, Z) \rightarrow works\_in(X, Y) \\
 & works\_in(X, Y) \rightarrow group(Y)
 \end{aligned}$$

In general, a set of CDs associated with an  $\text{ER}^+$  schema consists of *key* and *inclusion dependencies*, where the latter are a special case of TGDs [32].

Notice that it is possible to characterize the form of relational dependencies resulting from the translation of  $\text{ER}^+$  schemata into relational schemata. This characterization appears in [30]. Relational dependencies (key and inclusion dependencies) which are derived from  $\text{ER}^+$  schemata are said to be in *conceptual dependency form* (abbreviated as *CD-form*). It is also possible to show that a set of key and inclusion dependencies is associated with some  $\text{ER}^+$  schema if and only if it is in CD-form.

### 10.3.3 Relevant Sublanguages

418

Two subclasses of  $\text{ER}^+$  schemata that are of special interest are schemata where 419  
only the identity permutation can be used in is-a constraints between relationships, 420  
and where only relationships of bounded arity are allowed. 421

**Definition 10.2.** Consider a set  $\Sigma$  of CDs over a schema  $\mathcal{R}$ .  $\Sigma$  is a set of *identity* 422  
*permutation CDs (IPCDs)* if for each TGD of  $\Sigma$  of the form  $r_1(X_1, \dots, X_n) \rightarrow$  423  
 $r_2(X_{i_1}, \dots, X_{i_n})$ , where  $\{r_1, r_2\}$  encode relationships in the associated  $\text{ER}^+$  schema, 424  
it holds that  $[i_1, \dots, i_n] = \text{id}_n$ , where  $\text{id}_n$  is the identity permutation  $[1, \dots, n]$  of the 425  
set  $[n]$ .  $\Sigma$  is a set of *bounded arity CDs (BACDs)* if all the predicates encoding 426  
relationships in the associated  $\text{ER}^+$  schema are of bounded arity. 427

Being able to encode  $\text{ER}^+$  schemata into relational ones with dependencies, in 428  
our study, we will exploit techniques used in the case of relational schemata with 429  
dependencies. Henceforth, when using the term TGD (resp., KD), unless stated 430  
otherwise, we shall refer to TGDs (resp., KDs) that are part of a set of CDs (the 431  
results of this chapter do not hold in general). 432

## 10.4 Separable Conceptual Dependencies

433

In this section, we introduce the novel class of *nonconflicting CDs*. This class 434  
ensures that the TGDs and the KDs do not interact, so that answers to queries over 435  
an  $\text{ER}^+$  schema can be computed by considering the TGDs only, and ignoring the 436  
KD, once it is known that the initial data are consistent with respect to the schema. 437  
This semantic property is known as *separability* [11, 26]. 438

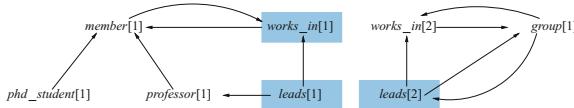
**Definition 10.3.** Consider a set  $\Sigma = \Sigma_T \cup \Sigma_K$  of CDs over a schema  $\mathcal{R}$ , where 439  
 $\Sigma_T$  are TGDs and  $\Sigma_K$  are KDs.  $\Sigma$  is *separable* if for every database  $D$  for  $\mathcal{R}$ , either 440  
 $\text{chase}(D, \Sigma)$  fails or, for every CQ  $q$  over  $\mathcal{R}$ ,  $\text{ans}(q, D, \Sigma) = \text{ans}(q, D, \Sigma_T)$ . 441

### 10.4.1 Definition of Nonconflicting CDs

442

Before syntactically defining nonconflicting CDs, we need some preliminary technical 443  
definitions. 444

Consider a set  $\Sigma = \Sigma_T \cup \Sigma_K$  of CDs over a schema  $\mathcal{R}$ , where  $\Sigma_T$  and  $\Sigma_K$  445  
are TGDs and KDs, respectively. The *CD-graph* of  $\Sigma$  is a multigraph  $\langle V, E, \lambda \rangle$ , 446  
where  $V$  is the node set,  $E$  is the edge set, and  $\lambda$  is a labeling function  $E \rightarrow \Sigma_T$ . 447  
The node set  $V$  is the set of positions of  $(f_e(\mathcal{R}) \cup f_r(\mathcal{R}))$ . For each TGD  $\sigma \in \Sigma_T$  448  
such that  $\{\text{pred}(\text{body}(\sigma)), \text{pred}(\text{head}(\sigma))\} \subseteq V$ , and for each universally quantified 449  
variable  $X$  that occurs in  $\text{body}(\sigma)$  at position  $\pi$  and in  $\text{head}(\sigma)$  at position  $\pi'$ , there 450  
exist: (1) an edge  $e$  from  $\pi$  to  $\pi'$  with  $\lambda(e) = \sigma$ , and (2) for each existentially 451



**Fig. 10.2** CD-graph for Example 10.5; k-nodes are *shaded* and special edges are represented using *broken lines*

quantified variable  $Y$  in  $\text{head}(\sigma)$  at position  $\pi''$ , a *special* edge  $e'$  from  $\pi$  to  $\pi''$  452 with  $\lambda(e') = \sigma$ . A node  $v = p[i] \in V$  is called *e-node* (resp., *r-node*) if  $p \in \mathbf{f}_e(\mathcal{R})$  453 (resp.,  $p \in \mathbf{f}_r(\mathcal{R})$ ). Moreover, if  $v$  is an r-node and the KD  $\text{key}(p) = \{i\}$  occurs in 454  $\Sigma_K$ , then  $v$  is called *k-node*. 455

Let  $G$  be the CD-graph of  $\Sigma$ . Consider an edge  $u \rightsquigarrow v$  in  $G$  which is labeled by the 456 TGD  $r_1(X_1, \dots, X_n) \rightarrow r_2(X_{j_1}, \dots, X_{j_n})$ , where  $\{r_1, r_2\} \subseteq \mathbf{f}_r(\mathcal{R})$  and  $[j_1, \dots, j_n]$  457 is a permutation of the set  $[n]$ . Intuitively, the permutation  $[j_1, \dots, j_n]$  indicates that 458 the  $j_i$ -th component of the relationship  $R_1$  is the  $i$ -th component of the relationship 459  $R_2$ . This fact can be represented by the bijective function  $f_{u \rightsquigarrow v} : [n] \rightarrow [n]$ , where 460 for each  $i \in [n]$ ,  $f_{u \rightsquigarrow v}(j_i) = i$ . Now, consider a cycle  $C = v_1^\rightsquigarrow v_2^\rightsquigarrow \dots^\rightsquigarrow v_m^\rightsquigarrow v_1$  of 461 only r-nodes in  $G$ . The permutation associated with  $C$ , denoted as  $\pi_G(C)$ , is defined 462 as the permutation  $g([1, \dots, n]) = [g(1), \dots, g(n)]$ , where  $g = f_{v_m^\rightsquigarrow v_1} \circ \dots \circ f_{v_2^\rightsquigarrow v_3} \circ$  463  $f_{v_1^\rightsquigarrow v_2}$ . 464

We are now ready to give the formal definition of nonconflicting CDs, which is 465 based on the notion of the CD-graph. 466

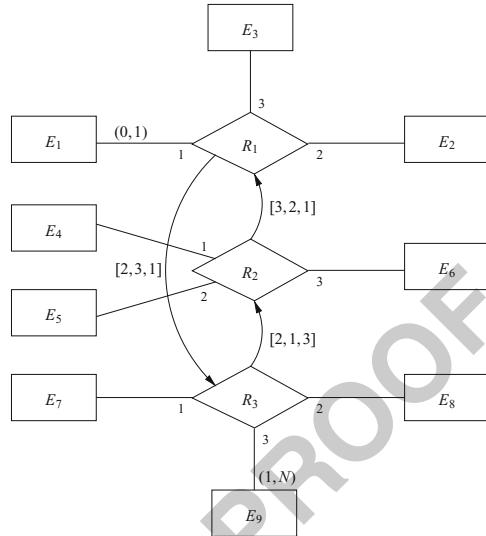
**Definition 10.4.** Consider a set  $\Sigma$  of CDs over a schema  $\mathcal{R}$ , and let  $G$  be the CD- 467 graph of  $\Sigma$ .  $\Sigma$  is *nonconflicting* if for each path  $v_1^\rightsquigarrow v_2^\rightsquigarrow \dots^\rightsquigarrow v_m$ , for  $m \geq 2$ , in  $G$  468 such that  $v_1$  is an e-node,  $v_2, \dots, v_{m-1}$  are r-nodes, and  $v_m$  is a k-node, the following 469 two conditions are satisfied: (1) for each cycle  $C$  of only r-nodes in  $G$  going through 470  $v_m$ ,  $\pi_G(C) = \text{id}_n$ , where  $n$  is the arity of the predicate of  $v_m$ , and (2) if  $m \geq 3$  and 471 the edge  $v_1^\rightsquigarrow v_2$  is nonspecial, then there exists a path of only r-nodes from  $v_m$  to  $v_2$ . 472

Observe that if only the identity permutation is allowed on is-a among relationships, then the first condition stated above is satisfied trivially. Thus, given a set of 473 IPCDs, to decide whether is nonconflicting, it suffices to check only the validity of 474 the second condition. 475

*Example 10.5.* Consider the  $\text{ER}^+$  schema  $\mathcal{C}$  given in Example 10.2; for simplicity, 477 we ignore the attributes. The set  $\Sigma$  of CDs associated with  $\mathcal{C}$  is given in Exam- 478 ple 10.4. The CD-graph of  $\Sigma$  is depicted in Fig. 10.2 (edge labels are omitted for 479 brevity). It is immediate to see that  $\Sigma$  is nonconflicting. ■ 480

In what follows, we give a more involved example of nonconflicting CDs, 481 where a cycle of only r-nodes going through a k-node occurs in the underlying 482 CD-graph, and also arbitrary permutations (other than the identity one) are used in 483 is-a constraints among relationships. 484

**Fig. 10.3** ER<sup>+</sup> schema for Example 10.6



*Example 10.6.* Consider the ER<sup>+</sup> schema  $\mathcal{C}$  depicted in Fig. 10.3; the formal definition to  $\mathcal{C}$  is omitted. The set  $\Sigma$  of CDs over  $\mathcal{R} = \{e_1, \dots, e_9, r_1, r_2, r_3\}$  associated with  $\mathcal{C}$  is the following:

$$\begin{aligned}
 \text{key}(r_1) &= \{1\}, \quad r_2(X, Y, Z) \rightarrow e_5(Y), \\
 e_9(X) \rightarrow \exists Y \exists Z r_3(Y, Z, X), \quad r_2(X, Y, Z) &\rightarrow e_6(Z), \\
 r_1(X, Y, Z) \rightarrow e_1(X), \quad r_2(X, Y, Z) \rightarrow r_1(Z, Y, X), \\
 r_1(X, Y, Z) \rightarrow e_2(Y), \quad r_3(X, Y, Z) \rightarrow e_7(X), \\
 r_1(X, Y, Z) \rightarrow e_3(Z), \quad r_3(X, Y, Z) \rightarrow e_8(Y), \\
 r_1(X, Y, Z) \rightarrow r_3(Y, Z, X), \quad r_3(X, Y, Z) \rightarrow e_9(Z), \\
 r_2(X, Y, Z) \rightarrow e_4(X), \quad r_3(X, Y, Z) \rightarrow r_2(Y, X, Z).
 \end{aligned} \tag{488}$$

Observe that the path  $e_9[1]^\sim r_3[3]^\sim r_2[3]^\sim r_1[1]$ , where the edge  $e_9[1]^\sim r_3[3]$  is nonspecial, occurs in the CD-graph  $G$  of  $\Sigma$ . Moreover, the edge  $r_1[1]^\sim r_3[3]$  occurs in  $G$ , and thus, the cycle  $C = r_1[1]^\sim r_3[3]^\sim r_2[3]^\sim r_1[1]$  occurs in  $G$ ; in fact,  $C$  is the only cycle of r-nodes in  $G$  going through the k-node  $r_1[1]$ . Clearly,

$$f_{e_1} = \begin{cases} 3 & \text{if } x = 1, \\ 1 & \text{if } x = 2, \\ 2 & \text{if } x = 3, \end{cases} \quad f_{e_2} = \begin{cases} 2 & \text{if } x = 1, \\ 1 & \text{if } x = 2, \\ 3 & \text{if } x = 3, \end{cases} \quad f_{e_3} = \begin{cases} 3 & \text{if } x = 1, \\ 2 & \text{if } x = 2, \\ 1 & \text{if } x = 3, \end{cases} \tag{493}$$

where  $e_1 = r_1[1]^\sim r_3[3]$ ,  $e_2 = r_3[3]^\sim r_2[3]$  and  $e_3 = r_2[3]^\sim r_1[1]$ . By defining  $g$  to be the composition  $f_{e_3} \circ f_{e_2} \circ f_{e_1}$ , we get that the permutation associated with  $C$  is  $\pi_G(C) = [g(1), g(2), g(3)] = [1, 2, 3] = \text{id}_3$ . Hence, the first condition of the

Definition 10.4 is satisfied. Due to the edge  $r_1[1] \curvearrowright r_3[3]$ , the second condition is also 497 satisfied. Thus,  $\Sigma$  is nonconflicting. ■ 498

The following example shows that during the construction of the chase of a 499 database with respect to a set of nonconflicting CDs, a KD may be violated which 500 implies that the KD chase rule must be applied. 501

*Example 10.7.* Consider the set  $\Sigma$  of nonconflicting CDs given in Example 10.4. 502 Let  $D = \{professor(p), leads(p, g)\}$ . During the construction of  $chase(D, \Sigma)$ , we 503 add the atoms  $member(p)$ ,  $works\_in(p, g)$ , and  $works\_in(p, z_1)$ , where  $z_1 \in \Gamma_N$ . 504 The KD  $key(works\_in) = \{1\}$  of  $\Sigma$  implies that  $p$  cannot participate more than 505 once in  $Works\_in$  as the first component. Thus, we deduce that  $z_1 = g$ . We must 506 therefore replace each occurrence of  $z_1$  with  $g$  in the part of the  $chase(D, \Sigma)$  507 constructed so far. ■ 508

### 10.4.2 Separable CDs vs. Nonconflicting CDs

509

In this subsection, we show that the nonconflicting condition is a sufficient and 510 necessary condition for separability. We first establish that every set of nonconflict- 511 ing CDs is separable. To this aim, we need an auxiliary technical lemma. 512

**Lemma 10.1.** Consider a database  $D$  for a schema  $\mathcal{R}$ , and a set  $\Sigma = \Sigma_T \cup \Sigma_K$  of 513 nonconflicting CDs over  $\mathcal{R}$ , where  $\Sigma_T$  are TGDs and  $\Sigma_K$  are KDs. If  $chase(D, \Sigma)$  514 does not fail, then there exists a homomorphism  $h$  that maps  $chase(D, \Sigma)$  to 515  $chase(D, \Sigma_T)$ . 516

*Proof.* We proceed by induction on the number of applications of the (TGD or KD) 517 chase rules in the construction of  $chase(D, \Sigma)$ . We need to show that, for each 518  $k \geq 0$ , there exists a homomorphism  $h_k$  that maps  $chase^{[k]}(D, \Sigma)$  to a set of atoms 519 of  $chase(D, \Sigma_T)$ . 520

BASE STEP. Clearly, for  $k = 0$ ,  $chase^{[0]}(D, \Sigma) = D \subseteq chase(D, \Sigma_T)$ . Therefore, 521 the claim holds trivially with  $h_0$  be the identity homomorphism on  $dom(D)$ . 522

INDUCTIVE STEP. By induction hypothesis, there exists a homomorphism  $h_{k-1}$  such 523 that  $h_{k-1}(chase^{[k-1]}(D, \Sigma)) \subseteq chase(D, \Sigma_T)$ . We proceed by considering the two 524 cases where the applied rule is either the TGD or the KD chase rule. 525

Suppose that we apply the TGD chase rule because the TGD  $\sigma = r(\mathbf{X}, \mathbf{Y}) \rightarrow$  526  $\exists \mathbf{Z} s(\mathbf{X}, \mathbf{Z})$  of  $\Sigma_T$  is violated. This implies that there exists a homomorphism  $\mu$  527 that maps  $r(\mathbf{X}, \mathbf{Y})$  to a set of atoms of  $chase^{[k-1]}(D, \Sigma)$ , and the atom  $a =$  528  $\mu'(s(\mathbf{X}, \mathbf{Z}))$  is obtained, where  $\mu'$  is an extension of  $\mu$  as in the TGD chase rule. 529 Clearly,  $chase^{[k]}(D, \Sigma) = chase^{[k-1]}(D, \Sigma) \cup \{a\}$ . Observe that the homomor- 530 phism  $\lambda = h_{k-1} \circ \mu$  maps  $r(\mathbf{X}, \mathbf{Y})$  to a set of atoms of  $chase(D, \Sigma_T)$ . Since 531  $chase(D, \Sigma_T) \models \Sigma_T$  it follows that there exists  $\lambda' \supseteq \lambda$  such that  $\lambda'(s(\mathbf{X}, \mathbf{Z})) \in$  532  $chase(D, \Sigma_T)$ . Denoting  $\mathbf{Z} = Z_1, \dots, Z_m$ , for  $m \geq 0$ , we define the sub- 533 stitution  $h_k = h_{k-1} \cup \{\mu'(Z_i) \rightarrow \lambda'(Z_i)\}_{i \in [m]}$ . Since none of the  $\mu'(Z_i)$  534

occurs in  $h_{k-1}$ , it follows that  $h_k$  is a well-defined substitution. Clearly,  $h_k(a) = h_k(\mu'(s(\mathbf{X}, \mathbf{Z}))) = s(h_{k-1}(\mu(\mathbf{X})), h_k(\mu'(\mathbf{Z}))) = s(\lambda(\mathbf{X}), \lambda'(\mathbf{Z})) = \lambda'(s(\mathbf{X}, \mathbf{Z})) \in chase(D, \Sigma_T)$ . Thus,  $h_k(chase^{[k]}(D, \Sigma)) \subseteq chase(D, \Sigma_T)$ , as needed.

Now, suppose that we apply the KD chase rule because the KD  $\kappa \in \Sigma_K$  is violated. We proceed by case analysis on the type of  $\kappa$ .

Assume first that  $\kappa$  is of the form  $key(a) = \{1\}$ , where  $a \in f_{ae}(\mathcal{R})$ . This implies that in  $chase^{[k-1]}(D, \Sigma)$ , there exist two atoms  $a = a(c_1, c_2)$  and  $b = a(c_1, c'_2)$ . If  $\{c_1, c_2, c'_2\} \subset \Gamma$ , then  $chase(D, \Sigma)$  fails and the claim holds trivially. The case where  $\{c_1, c_2, c'_2\} \subset \Gamma_N$ , and also the case where  $c_1 \in \Gamma$  and  $\{c_2, c'_2\} \subset \Gamma_N$  are excluded since, by definition of CDs,  $a$  and  $b$  must be the same atom which is a contradiction. Now, consider the case where  $\{c_1, c_2\} \subset \Gamma$  and  $c'_2 \in \Gamma_N$  (the case where  $\{c_1, c'_2\} \subset \Gamma$  and  $c_2 \in \Gamma_N$  is symmetric). Observe that, by definition of CDs,  $c'_2$  does not occur elsewhere in  $chase^{[k-1]}(D, \Sigma)$ . This implies that by applying  $\kappa$ , we just eliminate atom  $b$ , and thus  $chase^{[k]}(D, \Sigma) \subset chase^{[k-1]}(D, \Sigma)$ . Consequently,  $h_k = h_{k-1}$ .

Assume now that  $\kappa$  is of the form  $key(a) = [n]$ , where  $a \in f_{ar}(\mathcal{R})$ . This implies that in  $chase^{[k-1]}(D, \Sigma)$ , there exist two atoms  $a = a(c_1, \dots, c_n, c_{n+1})$  and  $b = a(c_1, \dots, c_n, c'_{n+1})$ . If  $\{c_1, \dots, c_{n+1}, c'_{n+1}\} \subset \Gamma$ , then  $chase(D, \Sigma)$  fails and the claim holds trivially. The cases where  $\{c_1, \dots, c_{n+1}, c'_{n+1}\} \subset \Gamma_N$ , or  $\{c_1, \dots, c_n\} \subset \Gamma$  and  $\{c_{n+1}, c'_{n+1}\} \subset \Gamma_N$ , or for some  $i \in [n]$ ,  $c_i \in \Gamma$  and  $\{c_1, \dots, c_{i-1}, c_{i+1}, \dots, c_{n+1}, c'_{n+1}\} \subset \Gamma_N$ , are excluded since, by definition of CDs,  $a$  and  $b$  must be the same atom which is a contradiction. Now, consider the case where  $\{c_1, \dots, c_{n+1}\} \subset \Gamma$  and  $c'_{n+1} \in \Gamma_N$  (the case where  $\{c_1, \dots, c_n, c'_{n+1}\} \subset \Gamma$ , and  $c_{n+1} \in \Gamma_N$  is symmetric). By definition of CDs,  $c'_{n+1}$  does not occur elsewhere in  $chase^{[k-1]}(D, \Sigma)$ . By applying  $\kappa$ , we just eliminate atom  $b$ , and hence,  $chase^{[k]}(D, \Sigma) \subset chase^{[k-1]}(D, \Sigma)$ . Therefore,  $h_k = h_{k-1}$ .

We continue to consider the case where  $\kappa$  is of the form  $key(r) = \{1\}$  (without loss of generality, we assume the first attribute to form the key). This implies that in  $chase^{[k-1]}(D, \Sigma)$ , we have the atoms  $a = r(c_1, c_2, \dots, c_n)$  and  $b = r(c_1, c'_2, \dots, c'_n)$ . If  $\{c_1, \dots, c_n, c'_2, \dots, c'_n\} \subset \Gamma$ , then  $chase(D, \Sigma)$  fails and the claim holds trivially. The case where  $\{c_1, \dots, c_{i-1}, c_{i+1}, \dots, c_n, c'_2, \dots, c'_{j-1}, c'_{j+1}, \dots, c'_n\} \subset \Gamma_N$  and  $\{c_i, c'_j\} \subset \Gamma$ , for some  $i, j \in \{2, \dots, n\}$ , is excluded since, by definition of CDs,  $a$  and  $b$  must be the same atom which is a contradiction. We define  $\mu = \{c'_i \rightarrow c_i\}_{2 \leq i \leq n}$ . During the application of the violated KD, we apply  $\mu$  to  $chase^{[k-1]}(D, \Sigma)$  and the atom  $b$  is eliminated (the case where  $\mu = \{c_i \rightarrow c'_i\}_{2 \leq i \leq n}$  and  $a$  is eliminated is symmetric). Consider an arbitrary atom  $c \in chase^{[k-1]}(D, \Sigma)$ , where  $dom(c) \cap \{c'_2, \dots, c'_n\}$  is not empty. We are going to show that  $h_{k-1}(\mu(c)) \in chase(D, \Sigma_T)$ . Suppose that the atom  $b$  is obtained during the chase due to the application of the TGD  $\sigma \in \Sigma_T$ . We proceed by case analysis on the type of  $\sigma$ .

Let  $\sigma = a(X_1, \dots, X_n, Y) \rightarrow r(X_1, \dots, X_n)$ , where  $a \in f_{ar}(\mathcal{R})$ . Observe that, by definition of CDs, the atoms generated during the chase due to the application of  $\sigma$  are necessarily of the form  $r(d_1, \dots, d_n)$ , where  $\{d_1, \dots, d_n\} \subset \Gamma$ . Therefore, this case is not possible since  $\{c'_2, \dots, c'_n\} \subset \Gamma_N$ .

Assume now that  $\sigma = e(X) \rightarrow \exists X_2 \dots \exists X_n r(X, X_2, \dots, X_n)$ , where  $e \in \mathbf{f}_e(\mathcal{R})$ .  
 Clearly, the labeled nulls  $c'_2, \dots, c'_n$  are created during the application of  $\sigma$ ,  
 that is,  $b$  is the atom at which  $c'_2, \dots, c'_n$  are invented during the construction  
 of the chase. Therefore, the atom  $c$  is obtained starting from  $b$ . Since, by  
 induction hypothesis,  $h_{k-1}(\mu(b)) \in \text{chase}(D, \Sigma_T)$ , it follows that  $h_{k-1}(\mu(c)) \in$   
 $\text{chase}(D, \Sigma_T)$ , as needed.  
584

Suppose now that  $\sigma = r'(X_1, \dots, X_n) \rightarrow r(X_{i_1}, \dots, X_{i_n})$ , where  $r' \in \mathbf{f}_r(\mathcal{R})$   
 and  $[i_1, \dots, i_n]$  is a permutation of  $[n]$ . Assume first that  $c$  is such that  $\text{pred}(c) \in$   
 $\mathbf{f}_r(\mathcal{R})$  and  $\text{dom}(c) = \{c_1, c'_2, \dots, c'_n\}$ . By definition of nonconflicting CDs,  
 the only way to obtain  $b$  during the chase is due to a path  $P$  of the form  
 $e[1]^\sim s_1[j_1]^\sim \dots^\sim s_m[j_m]^\sim r'[j_{m+1}]^\sim r[1]$ , where  $m \geq 0$ ,  $\{j_1, \dots, j_{m+1}\} \subseteq [n]$ ,  
 $e \in \mathbf{f}_e(\mathcal{R})$ , and  $\{s_1, \dots, s_m\} \subset \mathbf{f}_r(\mathcal{R})$ , in the CD-graph  $G$  of  $\Sigma$ . Note that  $e[1]$   
 is an e-node, and all the other nodes in  $P$  are r-nodes. Moreover, due to the  
 existence of  $\kappa$  in  $\Sigma_K$ ,  $r[1]$  is a k-node. In case that  $c$  is obtained starting from  $b$ ,  
 then  $h_{k-1}(\mu(c)) \in \text{chase}(D, \Sigma_T)$  since, by induction hypothesis,  $h_{k-1}(\mu(b)) \in$   
 $\text{chase}(D, \Sigma_T)$ . The interesting case is when  $c$  is generated before  $b$  due to the  
 existence of the path  $P$ . By definition of nonconflicting CDs, in  $G$ , we have a  
 path  $P'$  of only r-nodes from  $r[1]$  to  $s_1[j_1]$ . Furthermore, the permutation  $\pi_G(C)$ ,  
 where  $C$  is the cycle  $r[1]^\sim \dots^\sim s_1[j_1]^\sim \dots^\sim s_m[j_m]^\sim r'[j_{m+1}]^\sim r[1]$  in  $G$ , is  
 equal to  $\text{id}_n$ . Since, by induction hypothesis,  $h_{k-1}(\mu(b)) \in \text{chase}(D, \Sigma_T)$ , we get  
 that  $h_{k-1}(\mu(c)) \in \text{chase}(D, \Sigma_T)$ . Finally, suppose that  $c$  is such that  $\text{pred}(c) \in$   
 $\mathbf{f}_e(\mathcal{R}) \cup \mathbf{f}_{ae}(\mathcal{R}) \cup \mathbf{f}_{ar}(\mathcal{R})$ . It is clear that  $c$  is obtained starting from some atom  
 $d \in \text{chase}^{[k-1]}(D, \Sigma)$  such that  $\text{pred}(d) \in \mathbf{f}_r(\mathcal{R})$  and  $\text{dom}(d) = \{c_1, c'_2, \dots, c'_n\}$ .  
 Since  $h_{k-1}(\mu(d)) \in \text{chase}(D, \Sigma_T)$ , then  $h_{k-1}(\mu(c)) \in \text{chase}(D, \Sigma_T)$ , as needed.  
602

The desired homomorphism from  $\text{chase}(D, \Sigma)$  to  $\text{chase}(D, \Sigma_T)$  is eventually  
 $h = \bigcup_{k=0}^{\infty} h_k$ .  
□

By using Lemma 10.1, it is easy to establish separability of nonconflicting CDs.  
603

**Theorem 10.2.** Consider a set  $\Sigma$  of CDs over a schema  $\mathcal{R}$ . If  $\Sigma$  is nonconflicting,  
 then it is separable.  
604

*Proof.* Let  $\Sigma = \Sigma_T \cup \Sigma_K$ , where  $\Sigma_T$  are TGDs and  $\Sigma_K$  are KDs, and let  $D$  be  
 a database for  $\mathcal{R}$ . Suppose that  $\text{chase}(D, \Sigma)$  does not fail (otherwise the claim  
 holds trivially). By Lemma 10.1, we get that there exists a homomorphism that  
 maps  $\text{chase}(D, \Sigma)$  to  $\text{chase}(D, \Sigma_T)$ . Moreover, since  $\text{chase}(D, \Sigma_T)$  is a universal  
 model of  $D$  with respect to  $\Sigma_T$ , and  $\text{chase}(D, \Sigma) \in \text{mods}(D, \Sigma_T)$ , there exists a  
 homomorphism that maps  $\text{chase}(D, \Sigma_T)$  to  $\text{chase}(D, \Sigma)$ . Therefore,  $\text{chase}(D, \Sigma)$   
 and  $\text{chase}(D, \Sigma_T)$  are homomorphically equivalent, which implies that for each CQ  
 $q$  over  $\mathcal{R}$ ,  $q(\text{chase}(D, \Sigma)) = q(\text{chase}(D, \Sigma_T))$ . The claim follows by Theorem 10.1.  
605

□

Interestingly, for CDs, the property of being nonconflicting, it is not only suffi-  
 cient for separability, but it is also necessary. This way, we precisely characterize  
 the class of separable CDs by means of a graph-based syntactic condition.  
606  
607  
608

**Theorem 10.3.** Consider a set  $\Sigma$  of CDs over a schema  $\mathcal{R}$ . If  $\Sigma$  is not nonconflicting, then it is not separable. 609  
610

*Proof.* Let  $\Sigma = \Sigma_T \cup \Sigma_K$ , where  $\Sigma_T$  are TGDs and  $\Sigma_K$  are KDS. We prove this 611 result by exhibiting a database  $D$  such that  $chase(D, \Sigma)$  does not fail, and show that 612 there exists a Boolean CQ  $q$  such that  $\langle \rangle \in ans(q, D, \Sigma)$  but  $\langle \rangle \notin ans(q, D, \Sigma_T)$ . 613 Let  $G$  be the CD-graph for  $\mathcal{R}$  and  $\Sigma$ . We proceed by considering the following two 614 cases. 615

Suppose that the first condition of the Definition 10.4 is violated. Thus, there 616 exists a path  $v_1^\curvearrowright v_2^\curvearrowright \dots^\curvearrowright v_m$ , for  $m \geq 2$ , in  $G$  as in the Definition 10.4 for which the 617 following holds. Let  $v_1 = e_1[1]$  and  $v_i = r_i[j_i]$ , for each  $i \in \{2, \dots, m\}$ . Assume 618 that  $r_m$  is a predicate of arity  $n$ . There exists a cycle  $C$  of only r-nodes in  $G$  going 619 through  $r_m[j_m]$  such that  $\pi_G(C) \neq id_n$ . Consider the database  $D = \{e_1(c)\}$ . The 620 arc  $e_1[1]^\curvearrowright r_2[j_2]$  is necessarily labeled by the TGD 621

$$e(X) \rightarrow \exists X_1 \dots \exists X_{j_2-1} \exists X_{j_2+1} \dots \exists X_n r_2(X_1, \dots, X_{j_2-1}, X, X_{j_2+1}, X_n). \quad 622$$

Thus, eventually the atom  $r_m(z_1, \dots, z_{j_m-1}, c, z_{j_m+1}, \dots, z_n)$  is obtained during the 623 chase, where, for each  $i \in \{1, \dots, j_m-1, j_m+1, \dots, n\}$ ,  $z_i \in \Gamma_N$ . Since 624 the cycle  $C$  exists in  $G$ ,  $r_m(w_1, \dots, w_{j_m-1}, c, w_{j_m+1}, \dots, w_n)$  is generated, where 625  $\{w_1, \dots, w_{j_m-1}, w_{j_m+1}, \dots, w_n\} = \{z_1, \dots, z_{j_m-1}, z_{j_m+1}, \dots, z_n\}$ . Clearly, after the 626 application of the KD key( $r_m$ ) =  $\{j_m\}$ , we get an atom  $a$  such that at least two 627 attributes of  $a$  have the same null. Note that it is not possible to get such an 628 atom during the construction of  $chase(D, \Sigma_T)$ . Therefore, we can construct an 629 atomic Boolean CQ  $q$ , i.e., with just one atom in its body, for which there exists a 630 homomorphism  $h$  such that  $h(body(q)) = a$ , and thus  $h(body(q)) \in chase(D, \Sigma)$ , 631 but there is not a homomorphism  $h'$  such that  $h'(body(q)) \in chase(D, \Sigma_T)$ . Hence, 632  $\langle \rangle \in ans(q, D, \Sigma)$  but  $\langle \rangle \notin ans(q, D, \Sigma_T)$ . Obviously, since in  $dom(D)$  we have 633 just one constant of  $\Gamma$ , the chase does not fail. 634

Now, suppose that the second condition of the Definition 10.4 is violated. 635 This implies that there exists a path  $v_1^\curvearrowright v_2^\curvearrowright \dots^\curvearrowright v_m$ , for  $m \geq 3$ , in  $G$  as in 636 the Definition 10.4, but there is no path of only r-nodes from  $v_m$  to  $v_2$ . Assume 637 that  $v_1 = e_1[1]$  and  $v_i = r_i[j_i]$ , for each  $i \in \{2, \dots, m\}$ . Consider the database 638  $D = \{e_1(c), r_m(c, \dots, c)\}$ . The arc  $e[1]^\curvearrowright r_2[j_2]$  is necessarily labeled by the TGD 639

$$e(X) \rightarrow \exists X_1 \dots \exists X_{j_2-1} \exists X_{j_2+1} \dots \exists X_n r_2(X_1, \dots, X_{j_2-1}, X, X_{j_2+1}, X_n). \quad 640$$

Hence,  $r_2(z_1, \dots, z_{j_2-1}, c, z_{j_2+1}, \dots, z_n)$  is obtained during the chase, where, for 641 each  $i \in \{1, \dots, j_2-1, j_2+1, \dots, n\}$ ,  $z_i \in \Gamma_N$ . Eventually, due to the path 642  $v_2^\curvearrowright \dots^\curvearrowright v_m$ , the atom  $r_m(w_1, \dots, w_{j_m-1}, c, w_{j_m+1}, \dots, w_n)$  is generated, where 643  $\{w_1, \dots, w_{j_m-1}, w_{j_m+1}, \dots, w_n\} = \{z_1, \dots, z_{j_2-1}, z_{j_2+1}, \dots, z_n\}$ . Since  $v_m$  is a 644 k-node, we replace  $w_k$  with  $c$ , for each  $k \in \{1, \dots, j_m-1, j_m+1, \dots, n\}$ . Thus, 645 we get (among others) the atom  $r_2(c, \dots, c)$ . Instead, in  $chase(D, \Sigma_T)$ , the atom 646  $r_2(z_1, \dots, z_{j_2-1}, c, z_{j_2+1}, \dots, z_n)$  remains in place, and there is no way to obtain the 647 atom  $r_2(c, \dots, c)$  due to the absence of a path of only r-nodes from  $v_m$  to  $v_2$  in  $G$ . 648

Now, let us define the atomic Boolean CQ  $q = p \leftarrow r_2(c, \dots, c)$ . It is easy to see that  $\langle \rangle \in ans(q, D, \Sigma)$  but  $\langle \rangle \notin ans(q, D, \Sigma_T)$ . Finally, since we have just one constant of  $\Gamma$  in  $dom(D)$ , there is no chase failure.  $\square$

It is straightforward to see that Theorem 10.2 holds for IPCDs and BACDs, 641 since these two classes are special cases of CDs. Moreover, for both IPCDs and 642 BACDs, the property of being nonconflicting it is necessary for separability; the 643 proof of this result is similar to that of Theorem 10.3. The following corollary 644 follows immediately. 645

**Corollary 10.1.** Consider a set  $\Sigma$  of IPCDs or BACDs over a schema  $\mathcal{R}$ .  $\Sigma$  is 646 separable if and only if it is nonconflicting. 647

## 10.5 Query Answering Under Nonconflicting CDs

648

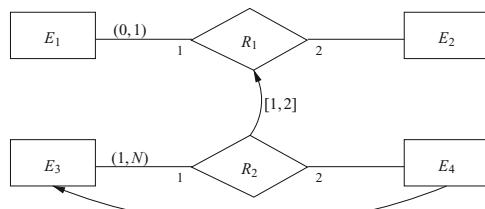
Let us now investigate the data and combined complexity of the (decision) problem 649 of conjunctive query answering under nonconflicting CDs (Fig. 10.4). 650

### 10.5.1 Data Complexity

651

As we shall see, once we know that the chase does not fail, the problem under 652 consideration is in the highly tractable class  $AC_0$  in data complexity. Before 653 we proceed further, let us recall the semantic notion of *first-order rewritability* 654 introduced in the context of description logics [5]. 655

A class  $\mathfrak{C}$  of dependencies is *first-order rewritable*, henceforth abbreviated as 656 *FO-rewritable*, if for every CQ  $q/n$ , and for every set  $\Sigma$  of dependencies in  $\mathfrak{C}$ , it 657 is possible to construct a (finite) first-order query  $q_\Sigma$  such that, for every database 658  $D$  and an  $n$ -tuple  $\mathbf{t} \in \Gamma^n$ ,  $\mathbf{t} \in ans(q, D, \Sigma)$  if and only if  $\mathbf{t} \in q_\Sigma(D)$ . It is well 659 known that evaluating first-order queries is in the highly tractable class  $AC_0$  in data 660 complexity [33]. Recall that  $AC_0$  is the complexity class of recognizing words in 661 languages defined by constant-depth Boolean circuits with an (unlimited fan-in) 662 AND and OR gates (see, e.g., [34]). 663



**Fig. 10.4** ER<sup>+</sup> schema for the proof of Theorem 10.5

**Theorem 10.4.** Consider a CQ  $q/n$  over a schema  $\mathcal{R}$ , a database  $D$  for  $\mathcal{R}$ , a set  $\Sigma$  of nonconflicting CDs over  $\mathcal{R}$ , and an  $n$ -tuple  $\mathbf{t} \in \Gamma^n$ . If  $\text{chase}(D, \Sigma)$  does not fail, then deciding whether  $\mathbf{t} \in \text{ans}(q, D, \Sigma)$  is in  $\text{AC}_0$  in data complexity. 664  
665  
666

*Proof.* Let  $\Sigma = \Sigma_T \cup \Sigma_K$ , where  $\Sigma_T$  are TGDs and  $\Sigma_K$  are KDs. Theorem 10.2 implies that  $\text{ans}(q, D, \Sigma)$  and  $\text{ans}(q, D, \Sigma_T)$  coincide. Thus, the problem whether  $\mathbf{t} \in \text{ans}(q, D, \Sigma)$  is equivalent to the problem whether  $\mathbf{t} \in \text{ans}(q, D, \Sigma_T)$ . Recall that  $\Sigma_T$  is a set of inclusion dependencies. It is well known that the class of inclusion dependencies is FO rewritable [11], and the claim follows.  $\square$

Since IPCDs and BACDs are special cases of CDs, it is obvious that the above result holds also for nonconflicting IPCDs and nonconflicting BACDs. Notice that Theorem 10.4 does not give the exact upper bound for the data complexity of the problem under consideration. This is because we assume that the chase does not fail. The data complexity of the problem whether the chase fails will be studied in Sect. 10.5.3. 667  
668  
669  
670  
671  
672

Interestingly, general CDs are not FO rewritable; the proof of this result is similar to a non-FO-rewritability proof given in [35]. 673  
674

**Theorem 10.5.** The class of CDs is not FO rewritable. 675

*Proof.* This result can be established by exhibiting a Boolean CQ  $q$  over a schema  $\mathcal{R}$ , a database  $D$  for  $\mathcal{R}$ , and a set  $\Sigma$  of CDs over  $\mathcal{R}$  for which there is no a (finite) first-order query  $q_\Sigma$  such that  $\langle \rangle \in \text{ans}(q, D, \Sigma)$  if and only if  $\langle \rangle \in q_\Sigma(D)$ .  $\square$

## 10.5.2 Combined Complexity

676

We now focus on the combined complexity of the CQ answering problem under nonconflicting CDs. First we show that, providing that the chase does not fail, the problem is in PSPACE in general, and is in NP in the case of IPCDs and BACDs. 677  
678  
679

**Theorem 10.6.** Consider a CQ  $q/n$  over a schema  $\mathcal{R}$ , a database  $D$  for  $\mathcal{R}$ , a set  $\Sigma$  of nonconflicting CDs (resp., IPCDs or BACDs) over  $\mathcal{R}$ , and an  $n$ -tuple  $\mathbf{t} \in \Gamma^n$ . If  $\text{chase}(D, \Sigma)$  does not fail, then the problem whether  $\mathbf{t} \in \text{ans}(q, D, \Sigma)$  is in PSPACE (resp., NP) in combined complexity. 680  
681  
682  
683

*Proof.* Let  $\Sigma = \Sigma_T \cup \Sigma_K$ , where  $\Sigma_T$  are TGDs and  $\Sigma_K$  are KDs. Since  $\text{chase}(D, \Sigma)$  does not fail, Theorem 10.2 implies that our problem is equivalent to the problem whether  $\mathbf{t} \in \text{ans}(q, D, \Sigma_T)$ . Recall that  $\Sigma_T$  is a set of inclusion dependencies. Moreover, observe that if  $\Sigma$  is a set of BACDs, then the maximum arity over all predicates of  $\mathcal{R}$  is bounded by a constant. The claim for nonconflicting CDs and nonconflicting BACDs follows from the fact that CQ answering under inclusion dependencies is in PSPACE in general, and in NP in the case of bounded arity [8]. 684  
685  
686  
687  
688  
689  
690  
691

The NP upper bound in the case of nonconflicting IPCDs does not follow immediately from the fact that CQ answering under inclusion dependencies, in the

case of bounded arity, is in NP since we have to deal with predicates of unbounded arity. It is possible to show that under nonconflicting IPCDs, for query answering purposes, it suffices to consider an initial finite part of the chase whose depth is polynomial with respect to  $q$  and  $\Sigma$ . This fact allows us to exhibit a nondeterministic PTIME algorithm which decides whether  $\mathbf{t} \in \text{chase}(D, \Sigma_T)$ . For the formal proof, we refer the reader to [36].  $\square$

As for Theorem 10.4, it is important to say that Theorem 10.6 does not provide the exact upper bound for the combined complexity of the problem under consideration, since we assume that the chase does not fail. The combined complexity of the problem whether the chase fails will be studied in the next subsection. 692  
693  
694  
695

Let us now establish the desired lower bounds for CQ answering under nonconflicting CDs (resp., BACDs, IPCDs). A useful decision problem that we are going to employ in the proof of the following result is the *finite function generation* (FFG) problem introduced and studied by Kozen in [37]. Consider a finite set of functions  $F \cup \{f\}$  from a set  $S$  to itself. The question is whether we can obtain  $f$  by composing functions of  $F$ . This problem is PSPACE hard, even in the case of bijective functions. 696  
697  
698  
699  
700  
701  
702

**Theorem 10.7.** *CQ-Ans under nonconflicting CDs is PSPACE hard in combined complexity.* 703  
704

*Proof.* The proof is by reduction of FFG. Consider a set  $F \cup \{f\}$  of bijective functions from a set  $S$  to itself. Suppose that  $F = \{f_1, \dots, f_m\}$ , for  $m \geq 1$ . Moreover, without loss of generality, assume that  $S = [n]$ , for some integer  $n \geq 2$ . 705  
706  
707

Let  $q$  be the Boolean CQ  $p \leftarrow r(c_{f(1)}, \dots, c_{f(n)})$ ,  $D = \{r(c_1, \dots, c_n)\}$ , where  $\{c_1, \dots, c_n\} \in \Gamma^n$ , and  $\Sigma$  be the set of CDs associated to the schema 708  
709

entity  $E$   
relationship  $R$  among  $\underbrace{E, \dots, E}_n$   
isa:  $R[f_1(1), \dots, f_1(n)], \dots, R[f_m(1), \dots, f_m(n)]$ . 710  
711  
712

Clearly,  $\Sigma$  is a set of nonconflicting CDs since in the CD-graph of  $\Sigma$ , there is no k-node. It is straightforward to see that  $\langle \rangle \in \text{ans}(q, D, \Sigma)$  if and only if  $f$  can be obtained by composing functions of  $F$ . This completes the proof.  $\square$

We conclude this subsection by showing that in the case of nonconflicting IPCDs and nonconflicting BACDs, CQ answering is NP hard. 713  
714

**Theorem 10.8.** *CQ-Ans under nonconflicting IPCDs and nonconflicting BACDs is NP hard in combined complexity.* 715  
716

*Proof.* It is well known that the problem of CQ answering under no dependencies, even for a relational schema with fixed arity, is NP hard in combined complexity [10]. The desired lower bounds can be easily established by providing a reduction of the above problem.  $\square$

### 10.5.3 Chase Failure

717

In this subsection, we show, by exploiting a technique proposed in [26], that the problem whether the chase fails is tantamount to the CQ answering problem (providing that the chase does not fail). In what follows, we are going to use the notion of union of CQs. A *union of conjunctive queries*  $Q$  of arity  $n$  is a set of CQs, where each  $q \in Q$  has the same arity  $n$  and uses the same predicate symbol in the head. The answer to  $Q$  over a database  $D$  is defined as the set  $Q(D) = \{\mathbf{t} \mid \text{there exists } q \in Q \text{ such that } \mathbf{t} \in q(D)\}$ . 724

**Lemma 10.2.** Consider a database  $D$  for  $\mathcal{R}$ , and a set  $\Sigma = \Sigma_T \cup \Sigma_K$  over  $\mathcal{R}$ , where  $\Sigma_T$  are TGDs and  $\Sigma_K$  are KDs. If both  $\mathcal{R}$  and  $\Sigma$  are arbitrary (resp., fixed), then we can construct in PTIME (resp., in AC<sub>0</sub>) a database  $D'$  and a union of Boolean CQs  $Q$  such that  $\text{chase}(D, \Sigma)$  fails if and only if  $\langle \rangle \in Q(\text{chase}(D', \Sigma_T))$ . 728

*Proof.* Let  $D'$  be the database obtained from  $D$  by adding a fact  $\text{neq}(c_1, c_2)$ , for each pair of distinct constants  $c_1$  and  $c_2$  that appear in  $\text{dom}(D)$ , where  $\text{neq}$  is an auxiliary predicate symbol not occurring in  $\mathcal{R}$ . The union of Boolean CQs  $Q$  is constructed as follows: for every KD  $\text{key}(r) = [m] \in \Sigma_K$ , where  $r/n \in \mathcal{R}$  (without loss of generality we assume the first  $m < n$  attributes to form the key), and for each  $j \in \{m+1, \dots, n\}$ , we add to  $Q$  the Boolean CQ  $p \leftarrow r(\mathbf{X}, Y_{m+1}, \dots, Y_n), r(\mathbf{X}, Z_{m+1}, \dots, Z_n), \text{neq}(Y_j, Z_j)$ , where  $\mathbf{X} = X_1, \dots, X_m$ . 735

In the case where both  $\mathcal{R}$  and  $\Sigma$  are arbitrary, then  $D'$  and  $Q$  can be constructed in polynomial time. In particular, the number of atoms that we add in  $D$  is less than  $(|\text{dom}(D)|)^2$ , while the number of Boolean CQs that we construct is  $|\Sigma_K|$ . Now, in the case where both  $\mathcal{R}$  and  $\Sigma$  are fixed, the required atoms can be obtained by evaluating the first-order query 740

$$\begin{aligned} \text{const}(X_i) &\leftarrow \bigvee_{r/n \in \mathcal{R}} \bigvee_{i \in [n]} \exists X_1 \dots \exists X_{i-1} \exists X_{i+1} \dots \exists X_n r(X_1, \dots, X_n), \\ \text{neq}(X, Y) &\leftarrow \text{const}(X) \wedge \text{const}(Y) \wedge X \neq Y, \end{aligned} \quad 741$$

over the database  $D$ , which is feasible in AC<sub>0</sub> in data complexity [33]. The above first-order query depends only on the schema  $\mathcal{R}$ , and thus can be constructed in constant time. Obviously,  $Q$  can be constructed in constant time. It is not difficult to show that  $\text{chase}(D, \Sigma)$  fails if and only if  $\langle \rangle \in Q(\text{chase}(D', \Sigma_T))$ .  $\square$

Notice that in the proof of the above result, the fact that the TGDs and the KDs are part of a set of CDs is not needed. This implies that the above lemma holds for arbitrary TGDs and KDs. By exploiting Lemma 10.2 and Theorems 10.4, 10.6–10.8, it is not difficult to establish the exact data and combined complexity of CQ answering. 746

**Corollary 10.2.** CQ-Ans under nonconflicting CDs is in AC<sub>0</sub> in data complexity, and PSPACE complete in combined complexity. Also, CQ-Ans under nonconflicting IPCDs and BACDs is NP complete in combined complexity. 747 748 749

## 10.6 Adding Negative Constraints

750

Negative constraints (NCs) of the form  $\varphi(\mathbf{X}) \rightarrow \perp$  (see Sect. 10.2.3) can be used 751  
 in order to express several relevant constructs in  $\text{ER}^+$  schemata, e.g., disjointness 752  
 between entities and relationships, and nonparticipation of entities to relationships, 753  
 but also more general ones. The new conceptual model which arises is called  $\text{ER}_\perp^+$ . 754  
 In fact, an  $\text{ER}_\perp^+$  schema  $\mathcal{C}$  is an  $\text{ER}^+$  schema with an additional set  $\Sigma_\perp$  of NCs over 755  
 $\mathcal{R}$ , where  $\mathcal{R}$  is the relational schema associated with  $\mathcal{C}$ . 756

*Example 10.8.* Consider the  $\text{ER}^+$  schema  $\mathcal{C}$  obtained from the one in Example 10.2 757  
 (see Fig. 10.1) by adding the entity *Pension\_scheme* and a relationship *Enrolled* 758  
 among *Member* and *Pension\_scheme*, without any cardinality constraints. The 759  
 fact that students and professors are disjoint sets can be represented with the NC 760  
 $\text{phd\_student}(X), \text{professor}(X) \rightarrow \perp$  (entity disjointness). Moreover, the fact that 761  
 a student cannot be enrolled in a pension scheme (i.e., it does not participate to 762  
 $\text{Enrolled}$  as the first component) can be represented by the NC  $\text{phd\_student}(X)$ , 763  
 $\text{enrolled}(X, Y) \rightarrow \perp$  (nonparticipation of an entity to a relationship). 764

A set of CDs and NCs is separable if the answers to queries can be computed by 765  
 considering the TGDs only, and ignoring the KDs and the NCs, once it is known 766  
 that the chase does not fail, and also that the chase satisfies the set of NCs. 767

**Definition 10.5.** Consider a set  $\Sigma = \Sigma_T \cup \Sigma_K$  of CDs over a schema  $\mathcal{R}$ , where  $\Sigma_T$  768  
 are TGDs and  $\Sigma_K$  are KDs, and a set  $\Sigma_\perp$  of NCs over  $\mathcal{R}$ .  $\Sigma \cup \Sigma_\perp$  is *separable* if for 769  
 every database  $D$  for  $\mathcal{R}$ , we have that either  $\text{chase}(D, \Sigma)$  fails or  $\text{chase}(D, \Sigma) \not\models \Sigma_\perp$  770  
 or, for every CQ  $q$  over  $\mathcal{R}$ ,  $\text{ans}(q, D, \Sigma) = \text{ans}(q, D, \Sigma_T)$ . 771

It is straightforward to see that given a set  $\Sigma$  of CDs and a set  $\Sigma_\perp$  of NCs, 772  
 the fact that  $\Sigma$  is nonconflicting is sufficient for separability of  $\Sigma \cup \Sigma_\perp$ ; this 773  
 follows immediately from Theorem 10.2. The question that comes up is whether 774  
 the nonconflicting property is also necessary for separability. It is not difficult to 775  
 show that there exists a set  $\Sigma$  of CDs and a set  $\Sigma_\perp$  of NCs such that  $\Sigma$  is not 776  
 nonconflicting, but  $\Sigma \cup \Sigma_\perp$  is separable. 777

Interestingly, if we consider only *strongly consistent* schemata [19], then the 778  
 nonconflicting property is also necessary for separability. An  $\text{ER}_\perp^+$  schema  $\mathcal{C}$ , where 779  
 $\Sigma$  is the CDs over  $\mathcal{R}$  associated with  $\mathcal{C}$  and  $\Sigma_\perp$  is the NCs over  $\mathcal{R}$  associated to  $\mathcal{C}$ , 780  
 is strongly consistent if there exists a (possibly infinite) instance  $I$  for  $\mathcal{R}$  such that 781  
 $I \models \Sigma \cup \Sigma_\perp$ , and for each  $e \in f_e(\mathcal{R})$ ,  $e(I) \neq \emptyset$ . It is possible to show that 782  
 the problem whether an  $\text{ER}_\perp^+$  schema  $\mathcal{C}$ , where the set of CDs associated with  $\mathcal{C}$  783  
 is nonconflicting, is strongly consistent is PSPACE complete. The formal proofs are 784  
 omitted and can be found in [36]. 785

We conclude this section by showing that the addition of NCs does not alter the 786  
 complexity of CQ answering under nonconflicting CDs. 787

**Theorem 10.9.** *CQ-Ans under nonconflicting CDs and NCs is in  $\text{AC}_0$  in data 788  
 complexity, and PSPACE complete in combined complexity. Also, CQ-Ans under 789  
 nonconflicting IPCDs and BACDs is NP complete in combined complexity.* 790

AQ4

*Proof.* Consider a CQ  $q/n$  over a schema  $\mathcal{R}$ ; a database  $D$  for  $\mathcal{R}$ ; a set  $\Sigma = \Sigma_T \cup \Sigma_K$  of nonconflicting CDs (resp., IPCDs, BACDs) over  $\mathcal{R}$ , where  $\Sigma_T$  are TGDs and  $\Sigma_K$  are KDs; a set  $\Sigma_\perp$  of NCs over  $\mathcal{R}$ ; and an  $n$ -tuple  $\mathbf{t} \in \Gamma^n$ . It holds that  $\mathbf{t} \in ans(q, D, \Sigma \cup \Sigma_\perp)$  if and only if  $chase(D, \Sigma)$  fails or  $chase(D, \Sigma) \not\models \Sigma_\perp$  or  $\mathbf{t} \in ans(q, D, \Sigma)$ . Therefore, we can decide whether  $\mathbf{t} \in ans(q, D, \Sigma \cup \Sigma_\perp)$  by applying the following simple algorithm; given an NC  $v = \varphi(\mathbf{X}) \rightarrow \perp$ , we denote by  $q_v$  the Boolean CQ  $p \leftarrow \varphi(\mathbf{X})$ :

1. Construct the database  $D'$  from  $D$  and the union of Boolean CQs  $Q$  from  $\Sigma_K$ , as described in the proof of Lemma 10.2.
2. If  $\langle \rangle \in Q(chase(D', \Sigma_T))$ , then *accept*.
3. If there exists  $v \in \Sigma_\perp$  such that  $\langle \rangle \in ans(q_v, D, \Sigma)$ , then *accept*.
4. If  $\mathbf{t} \in ans(q, D, \Sigma)$ , then *accept*; otherwise, *reject*.

Soundness and completeness of the above algorithm follows immediately from Lemma 10.2. Step 1 can be carried out in PTIME in general, and in AC<sub>0</sub> in case both  $\mathcal{R}$  and  $\Sigma$  are fixed (see Lemma 10.2). Since  $\Sigma$  is a set of nonconflicting CDs (resp., IPCDs, BACDs), the claim follows immediately from Corollary 10.2.  $\square$

## 10.7 Discussion

803

In this chapter, we have introduced an extension of Chen's Entity-Relationship model, that is, the ER<sup>+</sup> family. ER<sup>+</sup> is a very natural formalism for modeling data, as it builds upon the ER model, which has been used for decades in database design. We argue that the ER model is a powerful tool for ontological query answering, which at the same time is not awkward to understand and use; in particular, practitioners and even persons without any technical knowledge can profitably use ER (and its extensions) to express ontologies.

We have presented three ER<sup>+</sup> languages which, by means of mild restrictions specified by efficiently testable syntactic conditions, enjoy the separability property and first-order rewritability. The latter ensures not only that query answering under our languages is in the low complexity class AC<sub>0</sub>, but also that answering can be done by evaluating a suitable SQL query over the initial data. This opens the possibility of efficient implementation of query answering under nonconflicting ER<sup>+</sup> schemata on large datasets. Our study also pinpoints the complexity of query answering under our three nonconflicting ER<sup>+</sup> languages with respect to both data and combined complexity.

It is worth noticing that the problem we deal with in this chapter has several other applications. It is tightly related to the analogous problem of query answering on incomplete data under constraints [11, 12] and to semantic data integration [13]. Moreover, our problem of query answering is mutually reducible to the query containment problem (see, e.g., [14]); therefore, all our results carry on to the latter.

Finally, we remind the reader that the complexity results for most DL-Lite languages come, in fact, as special cases of our results on ER<sup>+</sup>. For instance,

nonconflicting  $\text{ER}_B^+$  with negative constraints is strictly more expressive than the languages  $\text{DL-Lite}_{\mathcal{F}}$  and  $\text{DL-Lite}_{\mathcal{R}}$  [5, 6]. This shows that if DL-Lite languages are useful and important for modeling database schemata and ontologies, then *a fortiori*,  $\text{ER}^+$  also is.

## References

831

- AQ5
1. Calì, A., Gottlob, G., Pieris, A.: Tractable query answering over conceptual schemata. Proceedings of ER, pp. 175–190 (2009) 832  
833
  2. Calì, A., Gottlob, G., Pieris, A.: Query answering under expressive Entity-Relationship schemata. Proceedings of ER, pp. 347–361 (2010) 834  
835
  3. Chen, P.P.: The entity-relationship model: Towards a unified view of data. ACM Trans. Database Syst. **1**, 9–36 (1976) 836  
837
  4. Calvanese, D., Lenzerini, M., Nardi, D.: Description logics for conceptual data modeling. Logics for Databases and Information Systems, pp. 229–263. Kluwer, Dordrecht (1998) 838  
839
  5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: the DL-Lite family. J. Autom. Reason. **39**, 385–429 (2007) 840  
841  
842
  6. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. Data Semant. **10**, 133–173 (2008) 843  
844
  7. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing implications of data dependencies. ACM Trans. Database Syst. **4**, 455–469 (1979) 845  
846
  8. Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. J. Comput. Syst. Sci. **28**, 167–189 (1984) 847  
848
  9. Calì, A., Lembo, D., Rosati, R.: Query rewriting and answering under constraints in data integration systems. Proceedings of IJCAI, pp. 16–21 (2003) 849  
850
  10. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. Proceedings of STOC, pp. 77–90 (1977) 851  
852
  11. Calì, A., Lembo, D., Rosati, R.: Decidability and complexity of query answering over inconsistent and incomplete databases. Proceedings of PODS, pp. 260–271 (2003) 853  
854
  12. van der Meyden, R.: Logical approaches to incomplete information: a survey. Logics for Databases and Information Systems, pp. 307–356. Kluwer, Dordrecht (1998) 855  
856
  13. Lenzerini, M.: Data integration: a theoretical perspective. Proceedings of PODS, pp. 233–246 (2002) 857  
858
  14. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: query answering under expressive relational constraints. Proceedings of KR, pp. 70–80 (2008) 859  
860
  15. Gogolla, M., Hohenstein, U.: Towards a semantic view of an extended Entity-Relationship model. ACM Trans. Database Syst. **16**, 369–416 (1991) 861  
862
  16. Markowitz, V.M., Shoshani, A.: Representing extended Entity-Relationship structures in relational databases: a modular approach. ACM Trans. Database Syst. **17**, 423–464 (1992) 863  
864
  17. Markowitz, V.M., Makowsky, J.A.: Identifying extended Entity-Relationship object structures in relational schemas. IEEE Trans. Software Eng. **16**, 777–790 (1990) 865  
866
  18. Battista, G.D., Lenzerini, M.: A deductive method for Entity-Relationship modeling. Proceedings of VLDB, pp. 13–21 (1989) 867  
868
  19. Artale, A., Calvanese, D., Kontchakov, R., Ryzhikov, V., Zakharyaschev, M.: Reasoning over extended ER models. Proceedings of ER, pp. 277–292 (2007) 869  
870
  20. Calì, A., Calvanese, D., Giacomo, G.D., Lenzerini, M.: Accessing data integration systems through conceptual schemas. Proceedings of ER, pp. 270–284 (2001) 871  
872
  21. Ortiz, M., Calvanese, D., Eiter, T.: Characterizing data complexity for conjunctive query answering in expressive description logics. Proceedings of AAAI (2006) 873  
874

10	The Return of the Entity-Relationship Model: Ontological Query Answering	279
----	--	-----

22.	Calvanese, D., Giacomo, G.D., Lenzerini, M.: On the decidability of query containment under constraints. Proceedings of PODS, pp. 149–158 (1998)	875 876
23.	Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. <i>Theor. Comput. Sci.</i> <b>336</b> , 89–124 (2005)	877 878
24.	Deutsch, A., Nash, A., Remmel, J.B.: The chase revisited. Proceedings of PODS, pp. 149–158 (2008)	879 880
25.	Marnette, B.: Generalized schema-mappings: from termination to tractability. Proceedings of PODS, pp. 13–22 (2009)	881 882
26.	Calì, A., Gottlob, G., Lukasiewicz, T.: A general Datalog-based framework for tractable query answering over ontologies. Proceedings of PODS, pp. 77–86 (2009)	883 884
27.	Calì, A., Gottlob, G., Pieris, A.: Advanced processing for ontological queries. <i>PVLDB</i> <b>3</b> , 554–565 (2010)	885 886
28.	Calì, A., Gottlob, G., Pieris, A.: Query answering under non-guarded rules in Datalog $+/-$ . Proceedings of RR, pp. 1–17 (2010)	887 888
29.	Calì, A., Gottlob, G., Kifer, M., Lukasiewicz, T., Pieris, A.: Ontological reasoning with F-Logic Lite and its extensions. Proceedings of AAAI (2010)	889 890
30.	Calì, A., Martinenghi, D.: Querying incomplete data over extended ER schemata. <i>TPLP</i> <b>10</b> , 291–329 (2010)	891 892
31.	Vardi, M.Y.: The complexity of relational query languages. Proceedings of STOC, pp. 137–146 (1982)	893 894
32.	Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading, MA (1995)	895 896
33.	Vardi, M.Y.: On the complexity of bounded-variable queries. Proceedings of PODS, pp. 266–276 (1995)	897 898
34.	Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, Reading, MA (1999)	899
35.	Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: What to ask to a peer: ontology-based query reformulation. Proceedings of KR, pp. 469–478 (2004)	900 901
36.	Calì, A., Gottlob, G., Pieris, A.: Ontological query answering under expressive Entity-Relationship schemata. Unpublished manuscript available from the authors (2011)	902 903
37.	Kozen, D.: Lower bounds for natural proof systems. Proceedings of FOCS, pp. 254–266 (1977)	904
AQ6	38. de Bruijn, J., Heymans, S.: Logical foundations of (e)RDF(s): complexity and reasoning. Proceedings of ISWC, pp. 86–99 (2007)	905 906
AQ7		

AUTHOR QUERIES

- AQ1. Kindly check the corresponding author.
- AQ2. Please check if edit to the sentence starting “If is applicable to...” is ok.
- AQ3. Kindly check whether the citation of Fig. 10.4 is OK as cited.
- AQ4. Please check the sentence starting “An ER<sup>+</sup>schema...” for completeness.
- AQ5. Ref. [1] is not cited in the text. Kindly cite the same or delete it from the referenc list.
- AQ6. Kindly update [36].
- AQ7. Ref. [38] is not cited in the text. Kindly cite the same or delete it from the reference list.

UNCORRECTED PROOF

# Chapter 11

## Linked Data Services and Semantics-Enabled Mashup

Devis Bianchini and Valeria De Antonellis

### 11.1 Introduction

The Web of Linked Data can be seen as a global database, where resources are identified through URIs, are self-described (by means of the URI dereferencing mechanism), and are globally connected through RDF links. According to the Linked Data perspective, research attention is progressively shifting from data organization and representation to linkage and composition of the huge amount of data available on the Web [1]. For example, at the time of this writing, the DBpedia knowledge base describes more than 3.5 million things, conceptualized through 672 million RDF triples, with 6.5 million external links into other RDF datasets. Useful applications have been provided for enabling people to browse this wealth of data, like Tabulator [7]. Other systems have been implemented to collect, index, and provide advanced searching facilities over the Web of Linked Data, such as Watson [15] and Sindice [39]. Besides these applications, domain-specific systems to gather and mash up Linked Data have been proposed, like DBpedia Mobile [4] and Revyu.com. DBpedia Mobile is a location-aware client for the semantic Web that can be used on an iPhone and other mobile devices. Based on the current GPS position of a mobile device, DBpedia Mobile renders a map indicating nearby locations from the DBpedia dataset. Starting from this map, the user can explore background information about his or her surroundings. Revyu.com is a Web site where you can review and rate whatever is possible to identify (through a URI) on the Web. Nevertheless, the potential advantages implicit in the Web of Linked Data are far from being fully exploited. Current applications hardly go beyond presenting together data gathered from different sources [10]. Recently, research on the Web

---

AQ1

D. Bianchini (✉) · V. De Antonellis

Department of Information Engineering, University of Brescia, via Branze 38,  
25123 – Brescia, Italy

e-mail: [bianchin@ing.unibs.it](mailto:bianchin@ing.unibs.it); [deantone@ing.unibs.it](mailto:deantone@ing.unibs.it)

of Linked Data has been devoted to the study of models and languages to add 28 functionalities to the Web of Linked Data by means of *Linked Data services*. As it is 29 known, Web services and semantic Web technologies have been widely addressed 30 and studied to bring into reality the principles of software reuse and interoperability, 31 although their uptake on a Web scale has been significantly less prominent than 32 initially foreseen [16]. This is due to several reasons: first of all, service-oriented 33 architectures have been mainly suited for enterprises and for the most part targeted 34 WSDL/SOAP-based Web services [21]. The world around services on the Web 35 has significantly evolved toward a proliferation of Web APIs, also called RESTful 36 services [45] when they conform to the REST architectural style [24], and of 37 RSS/Atom feeds. On the other hand, the complexity of semantic Web service mod- 38 eling solutions like OWL-S [36] and WSMO [11] requires the use of rich semantic 39 languages and complex reasoners, which hampered their significant uptake. For 40 the Web of Linked Data, different lightweight solutions have been proposed like 41 the WSMO-Lite language [47] for the semantic annotation of WSDL/SOAP-based 42 Web services and the combined use of MicroWSMO/hRESTS languages [32] for 43 annotating Web APIs, which do not present any structure and in most cases are 44 mainly described through Web pages in plain HTML for human consumption. 45

Tools have been created to specifically support users in the annotation of services 46 by leveraging the Web of Linked Data as a source of background knowledge. 47 Such applications, such as SWEET [35], which assist users in annotating HTML 48 descriptions of Web APIs, and SOWER,<sup>1</sup> for supporting the semantic annotation of 49 WSDL/SOAP-based Web services, rely on solutions like Watson [15] for browsing 50 the Web of Linked Data so that it is possible to identify suitable vocabularies 51 (e.g., eCl@ass, Good Relations [29], and FOAF) and use them for the annotation. 52 So-called Linked Data services can be used for processing the Web of Linked Data 53 and for producing Linked Data from the large body of information behind RESTful 54 services and WSDL services in legacy systems. The aim is to build a serving system 55 on top of Linked Data whereby people are enabled to collaboratively develop and 56 deploy applications by reusing existing components. 57

When the selection from the shelf of available components and their aggregation 58 are performed to build value-added, short-living, situational Web applications, 59 the term *mashup* is often used [5]. Mashup can be performed both for domain- 60 independent actions such as transforming data between different schemas or 61 determining how trustworthy is a portion of data and for domain-dependent 62 applications such as sending an SMS or booking a hotel online. Actually, users are 63 overwhelmed by the growing number of existing components, and this phenomenon 64 is also described as lost in hyperspace [18]. To facilitate mashup of components, an 65 information retrieval perspective, where the mashup designer has a specific query 66 in mind which he/she tries to answer, is often assumed. Approaches of this kind 67 are inspired or by Yahoo! Pipes, as, for example, semantic Web pipes [42], or by 68 the query-by-example paradigm, such as MashQL [30]. Note that these kinds of 69

---

<sup>1</sup><http://technologies.kmi.open.ac.uk/soa4all-studio/provisioning-platform/sower/>.

approaches hold the perspective of “data” mashup, while the vision of Linked Data services aims at extending the Web of Linked Data with “functionalities” that should be properly combined. According to this vision, we consider Web APIs on top of the Web of Linked Data and their composition or mashup.

In this chapter, we address the above-mentioned issues by proposing a new approach for “serving up” Linked Data by semantics-enabled mashup of existing Web APIs. A basic semantics-enabled model of Web APIs is proposed. The model has been designed (1) to support providers who publish new Web APIs used to access the Web of Linked Data, (2) to support the Web designer who aims at exploring and selecting available Web APIs to build or maintain a Web mashup, and (3) to make it available on top of the Web of Linked Data. Based on the proposed model, we define automated matching techniques apt to establish *semantic links* among Linked Web APIs according to properly defined criteria: (1) *functional similarity links*, set between APIs that provide similar functionalities and enable access to similar Linked Data, and (2) *functional coupling links*, set between APIs that can be wired to realize new, value-added Web mashups. Finally, we identify recurrent use cases to support the construction of Web mashups by means of Linked Web API composition. The analyzed use cases enable a proactive suggestion and interactive selection of available Linked Web APIs in order to support the browsing of available resources according to an exploratory perspective, that is, a step-by-step construction of the final mashup, which evolves depending on the available APIs and the associated links. The model and the techniques we propose adhere to the Linked Data principles for publishing resources on the Web in a meaningful way: (1) HTTP URIs are used to identify published resources (i.e., Web APIs); (2) useful (RDF) information are provided on the Web API description when someone looks up a Web API through its URI; and (3) functional similarity links and functional coupling links are set to relate Web APIs to each other, thus enabling easy development and sharing of new Web applications. Moreover, we rely on the Web of Linked Data for Web API semantic annotation, following the lesson learned for Linked Data services in [41], where Linked Data are used as source of knowledge.

The chapter is organized as follows. After describing a motivating example, we will discuss the state of the art on semantic Web services and Web of Linked Data and on existing mashup approaches. Therefore, we present the proposed approach for Linked Data services, the techniques, and the use cases to support the mashup construction. Finally, we close the chapter with the architecture of a system which implements the proposed metrics and use cases.

## 11.2 Motivating Example

107

Just consider Veruschka, a Web designer who has been charged by the organizers of a Michelangelo Antonioni’s movie festival to design a Web application which collects all the services related to the festival, such as search services to find movie

108

109

110

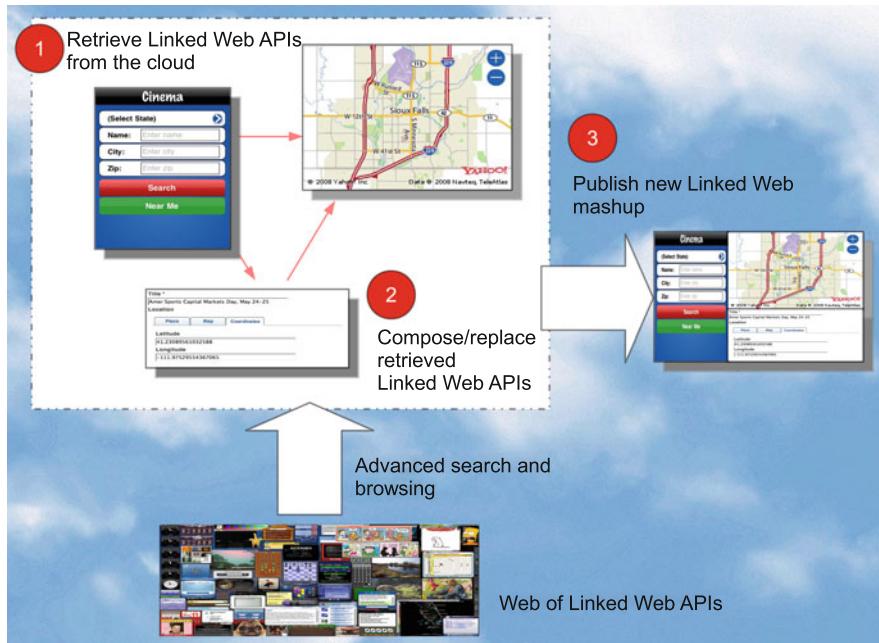
theaters which host the showing of the films, services to buy the tickets online, or services to reserve hotels in order to spend several days at the festival. This is an example of situational application, that is, an application targeted on the organizers' specific requirements and potentially useful for short periods of time (i.e., the interval of days in which the movie festival is held). The Web of Linked Data can be useful to collect knowledge on the considered topic and to expand this knowledge by means of RDF links. In a nutshell, this is the tenet of explorative search. For instance, the Web designer starts with a particular query in mind (e.g., Michelangelo Antonioni's movies) and linked metadata and knowledge can be suggested, such as information about Antonioni's preferred actress or a gallery of famous Antonioni's movie location, like Zabriskie Point, that were not in the Web designer's mind when he submitted the original query. Similarly, we rely on the same concepts to suggest Linked Data services used to access, manage, and transform the relevant information. Available services can be found among (1) Web APIs that enable to access contents coming from existing databases and legacy systems, such as Web APIs that provide information about reviews on movies (`Screeerat.com` or `Rottentomatoes.com`) or to obtain the addresses of movie theaters where a selected movie is shown, and (2) Web APIs and services that provide functionalities to process Linked Data, for example, to locate points on a map (`GoogleMaps APIs`) or to buy tickets online (e.g., `eBay APIs`). The Web designer decides to build his own Web application starting from available Web APIs instead of designing the application from scratch (that could be a nontrivial and time-consuming task, for instance, for what concerns displaying features on a map). On the other hand, browsing and selection of the right functionalities from a huge list of heterogeneous Web APIs is not an easy task if manually performed. See, for example, the scenario shown in Fig. 11.1. Firstly, the Web designer performs a generic search of available Web APIs by specifying some keywords, such as "movies." A list of available APIs whose descriptions match the specified keywords are proposed to him. Suggested Web APIs present the specified keywords in the API inputs/outputs, or have been associated with them as a related topic. By exploiting the Web of Linked Data, additional APIs can be suggested, annotated with related concepts, such as "movie theaters" or "film review." The Web designer keeps building the new Web application by adding new Linked Web APIs (e.g., the `GoogleMap APIs` to display the location of movie theaters) or by substituting already selected ones with new solutions found in the list. This procedure continues incrementally until the Web mashup is completed.

### 11.3 State of the Art

147

In this section, we will present the current background on Web services and Web API technologies and how they meet the semantic Web and the Web of Linked Data. Moreover, we will review the main efforts concerning support systems and technologies for data and service mashup in the context of Linked Data.

148  
149  
150  
151



**Fig. 11.1** The Linked Web mashup vision

### 11.3.1 Web Services and Web APIs

152

Web services have been proposed as a technology to increase software reuse 153 and interoperability across the Web. Web services are defined as loosely coupled 154 software components provided on the Web via open, platform-independent stan- 155 dards like XML, SOAP, and WSDL [21]. Web services can be composed to build 156 more complex, value-added distributed applications. Besides SOAP and WSDL, 157 several languages have been defined on the service-oriented architectures, such as 158 BPEL4WS [3] to model process-oriented service orchestration, WS-Security [37] 159 to apply security policies, WS-Interoperability [13] to improve interoperability 160 between vendor implementations, and so on. The stack of WS-\* specifications 161 is characterized by high complexity and is purely syntactic; therefore, service 162 discovery, composition, and monitoring are still a manual task. First attempts to 163 create a public registry of Web services, with limited, keyword-based searching 164 capabilities, proposed the Universal Business Registry within the Universal Descrip- 165 tion Discovery and Integration (UDDI) standard [28] and, more recently, the 166 ebXML Registry Information Model.<sup>2</sup> These proposals were discontinued in 2006. 167

<sup>2</sup><http://www.ebxml.org/>.

The main reason was that service-oriented architectures have been mainly designed 168 for enterprises, which have been always reluctant to publish their services on 169 the Web. Moreover, standards like UDDI do not support expressive queries [43]. 170 Finally, the Web service technology imposes that, for each functionality or software 171 component made available on the Web, the Web service provider must specify 172 complex WS-\* standards, thus preventing from the adoption of Web services on 173 a large scale. These features hindered the large application of Web services over the 174 Web. OPOSSum,<sup>3</sup> one of the most important Web service repositories on the Web, 175 lists few, toys Web service descriptions that are far from representing real software 176 components, while the largest list of publicly available Web services, accounting 177 for 25000 Web services with their documentation, is Seekda . com. Therefore, the 178 number of available Web services is considerably different from the huge amount 179 of currently available Web documents. 180

The advent and the diffusion of the so-called Web 2.0, which gives to the user a 181 leading role for Web content publication and for new Web application development, 182 influenced the proliferation of Web APIs, also called RESTful services [45] when 183 they conform to the REST architectural style [24]. Web APIs are characterized 184 by a very simple technological stack, based on HTTP, XML, and JSON, and the 185 extensive use of URIs. Nowadays, Web APIs are used to access large quantity of 186 data, like Flickr and Facebook, or to expose on the Web contents inside legacy 187 systems. In Programmableweb . com,<sup>4</sup> an online registry of mashups and APIs, 188 nearly 3,500 Web APIs and 5,900 mashups are listed at the time of this writing, 189 while Yahoo! Pipes contains over 20,000 user-defined components. Web APIs are 190 usually described using plain, unstructured HTML, although some efforts such 191 as WADL (Web application description language) [27] are being developed to be 192 the counterpart of the WSDL standard for WSDL/SOAP-based Web services. The 193 unstructured nature of Web API descriptions makes their automatic discovery a 194 difficult task, that is, currently mainly performed manually. In [25], a faceted classifi- 195 cation of unstructured Web APIs and a ranking algorithm to improve their retrieval 196 are proposed. The classification and searching solution is based on IR techniques. 197

### 11.3.2 Semantic Web Services

198

Semantic Web services have been proposed to apply the semantic Web vision [6] to 199 the world of Web services. We can define a semantic Web service as a Web service 200 whose functionality is further specified through a semantic annotation over a well- 201 defined ontology or semantic model. 202

**Definition 11.1 (Semantic Web Service).** We define a semantic Web service as a 203 pair  $\langle \mathcal{S}, \mathcal{A} \rangle$ , where  $\mathcal{S}$  is the Web service description and  $\mathcal{A}$  is the corresponding 204 semantic annotation expressed by an ontology or semantic model. 205

<sup>3</sup>OPOSSum - Online Portal for Semantic Services, <http://hnsn.inf-bb.uni-jena.de/opossum/>.

<sup>4</sup><http://www.programmableweb.com/>.

This definition has been meant for WSDL/SOAP-based Web services and in this context, a number of conceptual models have been proposed. Considering existing languages and service discovery infrastructures based on them,  $S$  mainly coincides with the Web service abstract interface contained in the WSDL specification. The semantic annotation  $\mathcal{A}$  ranges from complex, top-down solutions like the OWL-S [36] or the WSMO [11] ones to the bottom-up perspective of the SAWSDL [22] proposal. In particular, OWL-S defines an upper level service ontology, where a semantic Web service has three properties: it *presents* a *Service-Profile*, describing what the service does (i.e., the Web service interface), exploited for discovery purposes; it is *describedBy* a *ServiceModel*, describing how the service works; and it *supports* a *ServiceGrounding* for its invocation. WSMO proposes a semantic Web service model based on four core elements: (1) *ontologies*, to provide the terminology used by other WSMO elements in terms of machine-processable formal definitions; (2) *Web services*, to describe all the aspects of Web Services, including their *nonfunctional properties*, their *capabilities*, and their *interfaces*; (3) *goals*, to represent the user's requirements in terms of expected outputs and effects; and (4) *mediators*, to deal with interoperability problems between different WSMO elements. The Web service interfaces and goals are the elements involved in the discovery process. The SAWSDL proposal distinguished between the WSDL specification and the semantic model, introducing special extension attributes for semantic annotation of interface elements (e.g., operations, input/output messages). SAWSDL supports semantic annotation of WSDL and XML documents by means of three kinds of annotations, namely, *model reference*, *lifting schema mapping*, and *lowering schema mapping*, which allow pointing to semantic elements on the Web and specify data transformation from a syntactical representation to the semantic counterpart and vice versa. With respect to OWL-S/WSMO, by externalizing the semantic model, an agnostic perspective on the ontology representation language is taken, without constraining to a particular semantic model. Recently, efforts to ground OWL-S to SAWSDL and several tools and APIs to facilitate the use of these languages, hiding representation details, have been developed, and SAWSDL is gaining more and more popularity as semantic Web service description language. Such models are essentially incompatible to each other due to the adopted representation language and expressivity of the semantic model, thus raising interoperability issues. The World Wide Web Consortium (W3C) proposed a WSDL RDF mapping [31] to add WSDL data to the semantic Web. This mapping specification has two main parts: a WSDL ontology and a formal mapping from WSDL documents to RDF data. Since this representation is independent from the WSDL one, it is extensible and supports any mechanism to annotate Web services. Moreover, interoperability is achieved by merging different semantic Web data and allowing to read WSDL documents as RDF inputs. Nevertheless, the inefficiency of this representation presents relevant issues of scalability and management capabilities [17].

As stated above, the most part of semantic Web service approaches are focused on WSDL/SOAP-based Web services, while the attention is shifting toward Web APIs and RESTful services. Moreover, all these approaches are characterized by a considerable complexity in the conceptual model used for the semantic annotation

of services and Web APIs. As wished by [40], with the advent of the Web of Linked Data and with the new role assumed by the users that are recently referred to as *prosumers* joining the characteristics of providers and consumers, lightweight semantics is yielding significant benefits that justify the investment in annotating data and services. In this context, the Web of Linked Data can be assumed as a repository of knowledge to be used for supporting the annotation of Linked Data services. Tools such as SWEET [35] work in this direction. In particular, SWEET guides the providers to give a structured representation of their Web APIs (by using the hRESTS formalism) and add semantics by referencing concepts in publicly available domain ontologies through the MicrowSMO language [32]. Another related effort is the SA-REST language [33] that allows semantic annotation of RESTful service descriptions, and it is proposed in the context of semantic mashups. Data mediation between heterogeneous services is addressed by means of lowering and lifting schema mappings. In SA-REST, semantic annotation of RESTful services should be supplied by their providers. Similarly, for what concerns WSDL/SOAP-based services, WSMO-Lite [47] has been proposed to overcome some of the SAWSDL limitations that do not commit any semantic model. Basically, WSMO-Lite includes a lightweight RDFS ontology together with a methodology to express functional and nonfunctional semantics and an information model for WSDL services based on the SAWSDL model reference.

In order to provide a common vocabulary based on existing Web standards to semantically describe services and to enable their publication on the Web of Linked Data, [40] proposed the minimal service model (MSM), originally introduced together with hRESTS and WSMO-Lite. In a nutshell, the MSM is a simple RDFS built upon existing vocabularies, namely, SAWSDL, WSMO-Lite, and hRESTS, and captures the maximum common denominator between existing conceptual models for services. On top of the MSM, the iServe architecture is implemented [41], to publish semantic annotations of services as Linked Data and provide a SPARQL endpoint for executing advanced querying over the service annotations.

### 11.3.3 Computer-Aided Web API Mashup

280

Recent efforts to provide mashup platforms underline the importance of mitigating the burden of mashup composition that requires programming skills, the in-depth knowledge of inputs/outputs formats for manually implementing data mediation in the final composition, and the integration of different kinds of components (e.g., components with their own UI, WSDL/SOAP-based Web services, RESTful services, RSS/Atom feeds).

To better support mashup development, it is crucial to abstract from underlying heterogeneity. In [14], an abstract component model and a composition model are proposed, expressed by means of an XML-based language. In particular, components are defined as stateful elements, with events for state changes notification and operations for querying and modifying the state. A composition model is

also proposed, where components are coupled according to a publish/subscribe mechanism, where events raised from a component are associated with operations of other components in the final mashup application. Both the component model and the composition model are not based on any semantic characterization. In [2], a formal model based on datalog rules is proposed to capture all the aspects of a mashup component. Mashups are combined into patterns, and a notion of inheritance relationship between components is introduced. Nevertheless, the proposed formal model is quite complex, and it is difficult to adapt it to the Linked Data perspective, where the adoption of lightweight semantics techniques is of paramount importance.

Works [2, 14] do not provide environments with on-the-fly support for mashup composition. Mashup composition is manual, and no component selection strategies or metrics to support it is proposed. In [38], authors propose a novel application of semantic annotation together with a matching algorithm for finding sets of functionally equivalent components out of a large set of available non-Web-service-based components. The approach described in [26] addresses the problem of proactive suggestion of components to be combined in a mashup. The MATCHUP system is proposed to support a rapid, on-demand, and intuitive mashup development. This approach is based on the notion of autocompletion: when the designer selects a component (called *mashlet*), the system suggests other components to be connected on the basis of recurrent patterns of components in the repository. The MATCHUP system is based on the formal component model described in [2], therefore suffers from the limitations we underlined above. A tag-based navigation technique for composing data mashups to integrate data sources is proposed in [44]. MashMaker [20] suggests widgets that might assist in handling data currently managed by the mashup designer. For example, the tool might suggest adding “map” location or “distance” widgets if the designer currently views a list of addresses. The MashupAdvisor [19] system recommends a set of possible outputs for a specific mashup. Each specific output corresponds to some transformation of the data being manipulated by the mashup. After the user has selected one output, MashupAdvisor computes an extension of the mashup in order to achieve the selected output. A Web-based interface which supports mashup of semantic-enriched Web APIs is proposed in sMash [34]. Possible mashups are shown as a graph, where each vertex represents an API, and an edge between two APIs means that they are mashupable, that is, they can be used together in a mashup.

## 11.4 Linked Web APIs Model

327

A fundamental step for bridging Web APIs closer to the Web of Linked Data is their publication based on current best practices. Following the lesson learnt by the iServe platform [41], which proposes the use of Linked Data publication principles [10] to enrich the Web of Data with functionalities provided as OWL-S, SAWSDL, and RESTful services, we propose a conceptual model for semantically annotated

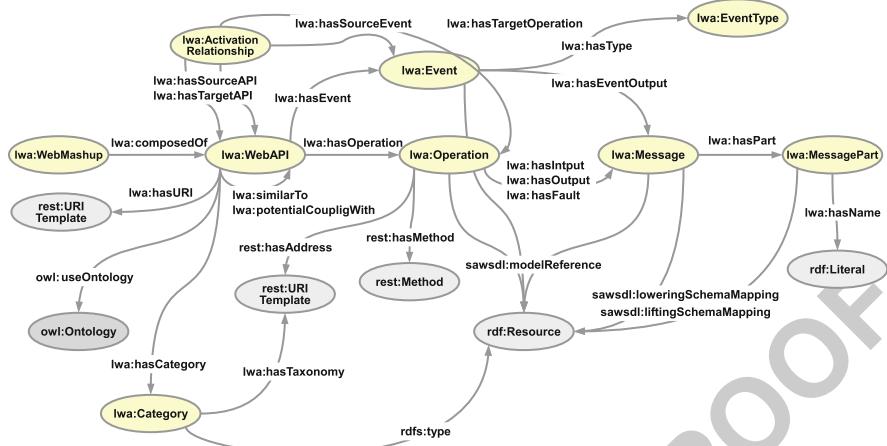
328

329

330

331

332



**Fig. 11.2** The semantics-enabled Web API conceptual model

Web APIs and Web mashups in order to link them to the Web of Data through the use of lightweight semantics. The conceptual model abstracts from implementation aspects and guides the publication of Web APIs within the Web of Linked Data through the application of matching techniques introduced in [9], which are based on the semantic annotations of model elements. Matching techniques support the identification of semantic links between APIs in the Web of Linked Data. We distinguish between two kinds of semantic links: *functional similarity links* between APIs which provide similar functionalities and *functional coupling links* between APIs which can be wired together in a Web mashup.

Basic elements of the semantics-enabled Web API model are shown in Fig. 11.2 and have been defined in the `lwa` namespace. We rely on existing vocabularies, namely SAWSDL, RDF(S), and hRESTS, identified with the `sawsdl`, `rdf/rdfs`, `rest` namespace prefixes, respectively. A Web API is uniquely identified by a URI. Moreover, a set of common elements can be identified in Web API descriptions: (1) *inputs* and *outputs*, represented through the `Message` construct, which in turn can be composed of several `MessageParts`; (2) *operations*, usually associated with buttons or links on the Web API graphical interface; and (3) *events*, to model user's interactions with the Web API interface. Operations are described by an address, which is built by concatenating the API URI with the operation name and is used to invoke the operation itself, and a method (e.g., classical POST and GET HTTP methods) for its invocation. Events are described by an event type (e.g., `onmouseover`, `onmouseclick`) and the outputs or arguments which are raised on event occurrence (for instance, a click on a map raises an event which contains the coordinates of the points which the mouse is positioned on). Message parts have a name that is modeled as an RDF literal. To add semantics to the Linked Web API model, we used the *model reference*, *lifting schema mapping*, and *lowering schema mapping* constructs proposed in the SAWSDL specification. Schema mappings are

used to provide grounding from the conceptual model to the concrete message formats (lowering schema mapping) and vice versa (lifting schema mapping).  
Model reference construct is used to associate operation names, inputs and outputs, and event outputs to concepts taken from the Web of Linked Data. Concepts are also identified by means of URIs as usual in the Web of Linked Data. Finally, APIs are classified with respect to categories (`hasCategory` construct) that are taken from standard taxonomies available on the Web (identified by a taxonomy URI). We do not commit a particular ontology formalism, but the wide range of ontologies and semantic data available on the Web of Linked Data is used as a source of knowledge. To this aim, techniques, formats, and tools for semantic annotation of Web APIs based on lightweight semantic models, like the SWEET tool proposed in [35], have been adapted to the proposed Web API model. The SWEET tool provides support in the identification of elements that characterize a Web API, which typically relies on nonstructured HTML documentation that makes difficult machine processing, to produce a description according to the hRESTS language. Annotation and classification of the Web APIs is performed according to the MicroWSMO [32] notation extended with semantically annotated events.

An event of a Web API can be connected to an operation of another Web API in a publish/subscribe-like mechanism. An event-operation pair is represented in the model through an activation relationship. For example, in Fig. 11.1, an activation relationship is shown from the cinema API to the map API: once a movie title is specified in the former, the locations of the projection rooms for the selected movie are shown on the map. An activation relationship is modeled with reference to the source and target Web APIs (`hasSourceAPI` and `hasTargetAPI` constructs), to the source event (`hasSourceEvent` construct), and to the target operation (`hasTargetOperation` construct).

Finally, Web APIs published on the Web of Linked Data are linked toward other APIs that provide similar functionalities (`similarTo` construct, which models the functional similarity link) or other APIs that could be coupled with the current one and included in the same Web mashup (`potentialCouplingWith` construct, which models the functional coupling link). Exploitation of these links will be shown in Sect. 11.6. Functional similarity links and functional coupling links aim at being the counterpart of RDF links of the Web of Linked Data. They could be set during Web API publication. In this way, the model we propose relies on the Linked Data principles [10] for Web API publication: (1) HTTP URIs are used to identify published resources (i.e., Web APIs); (2) useful (RDF) information are provided on the Web API description when someone looks up a Web API through its URI; (3) RDF statements (`similarTo` and `potentialCouplingWith` constructs) are used that link to other Web APIs that are related to each other (through functional similarity and functional coupling links).

Activation relationships, functional similarity links, and functional coupling links are based on the identification of IO correspondences across different Web APIs. In the next section, we describe matching techniques built on top of Linked Data to support the Web API provider and Web mashup designer during the identification of such correspondences.

## 11.5 Linked Web API Publication

405

As stated above, semantic characterization of Web API descriptions is performed 406 by relying on existing tools and techniques, such as SWEET. Nevertheless, pub- 407 lication of new Web APIs and new Web mashups on the Web of Linked Data 408 must be properly supported to help providers in identifying relationships with 409 existing resources. We distinguish among two situations: (a) publication of a new 410 semantically described Web API by a Web API provider and (b) publication of a 411 new Web mashup by a Web designer. 412

*Publication of a new semantically annotated Web API.* The Web API provider has 413 to identify related Web APIs that could be linked to the current one through 414 `similarTo` or `potentialCouplingWith` constructs. 415

*Publication of a new Web mashup.* The Web designer has to select available Web 416 APIs and properly set activation relationships between them. 417

We provide a set of matching techniques to support the Web API provider 418 and the Web designer to perform their tasks. These techniques are based on the 419 computation of a concept affinity  $CAff() \in [0..1]$  between pairs of concepts, used 420 in the semantic annotation of, respectively, (1) operations, (2) I/Os parameters, and 421 (3) event outputs and operation inputs. To this purpose, we rely on techniques such 422 as those extensively defined in [8]. Here, we simply state that  $CAff$  is based on both 423 a terminological (domain-independent) matching based on the use of WordNet [23] 424 and a semantic (domain dependent) matching based on the ontologies on the Web 425 of Linked Data, used as source of knowledge. 426

Given a new semantically annotated Web API description  $\mathcal{W}_i$ , another Web API 427  $\mathcal{W}_j$  is *functionally similar* to  $\mathcal{W}_i$  if (1) the  $\mathcal{W}_i$  and  $\mathcal{W}_j$  categories are compatible, 428 that is, at least one  $\mathcal{W}_j$  category is the same or less specific than one  $\mathcal{W}_i$  category, 429 and (2) the *functional similarity degree*  $Sim_{IO}(\mathcal{W}_i, \mathcal{W}_j)$  is equal to or greater than 430 a threshold  $\delta \in [0, 1]$  experimentally set. The functional similarity degree is defined 431 as follows. 432

**Definition 11.2 (Functional Similarity Degree).** The Functional Similarity 433 Degree  $Sim_{IO}(\mathcal{W}_i, \mathcal{W}_j) \in [0, 1]$  between two semantically annotated Web APIs 434  $\mathcal{W}_i$  and  $\mathcal{W}_j$  is evaluated as: 435

$$Sim_{IO}(\mathcal{W}_i, \mathcal{W}_j) = \frac{1}{3} \left[ \frac{\sum_{s,t} CAff(in_s, in_t)}{|IN(\mathcal{W}_j)|} + \frac{\sum_{h,k} CAff(out_h, out_k)}{|OUT(\mathcal{W}_i)|} \right. \\ \left. + \frac{\sum_{l,m} CAff(op_l, op_m)}{|OP(\mathcal{W}_i)|} \right] \quad (11.1)$$

where  $IN(\mathcal{W}_j)$  (resp.,  $OUT(\mathcal{W}_i)$ ) is the set of concepts used to annotate the 436 operation inputs of the  $\mathcal{W}_j$  Web API description (resp., the operation outputs of 437 the  $\mathcal{W}_i$  Web API description);  $OP(\mathcal{W}_i)$  is the set of concepts used to annotate 438

the operations of the  $\mathcal{W}_i$  Web API description;  $in_s \in IN(\mathcal{W}_i)$ ,  $in_t \in IN(\mathcal{W}_j)$ ,  $out_h \in OUT(\mathcal{W}_i)$ ,  $out_k \in OUT(\mathcal{W}_j)$ ,  $op_l \in OP(\mathcal{W}_i)$ ,  $op_m \in OP(\mathcal{W}_j)$ . 439  
440

The  $Sim_{IO}(\mathcal{W}_i, \mathcal{W}_j)$  formula measures how much  $\mathcal{W}_j$  provides at least the operations and outputs required in  $\mathcal{W}_i$ , no matter if  $\mathcal{W}_j$  provides additional operations and outputs. The building block of this expression is the Dice coefficient [46], used in information retrieval. When the provider publishes a new semantically annotated Web API description, a preliminary search of the available Web APIs on the Web of Linked Data is performed on the basis of Web API categories, and the functional similarity degree is computed. Those Web APIs which present a functional similarity degree value equal to or greater than the threshold  $\delta$  are proposed to the provider to set the `similarTo` link between them. 441  
442  
443  
444  
445  
446  
447  
448  
449

Similarly, we defined a procedure to assist the Web API provider for finding potentially coupled Web APIs on the Web of Linked Data. Given a new semantically annotated Web API description  $\mathcal{W}_i$ , another Web API  $\mathcal{W}_j$  can be *functionally coupled* with  $\mathcal{W}_i$  if the *functional coupling degree*  $Coupl_{IO}(\mathcal{W}_i, \mathcal{W}_j)$  is equal to or greater than a threshold  $\theta \in [0, 1]$  experimentally set. The functional coupling degree is based on the *event-operation coupling coefficient*  $Coupl_{EvOp}(ev_i, op_j)$ , evaluated as the average  $CAff()$  value between the event outputs of  $ev_i \in EV(\mathcal{W}_i)$  and the inputs of  $op_j \in OP_j$ , where  $EV(\mathcal{W}_i)$  is the set of events of the  $\mathcal{W}_i$  Web API description. 450  
451  
452  
453  
454  
455  
456  
457  
458

**Definition 11.3 (Event-Operation Coupling Degree).** The Event-Operation Coupling Degree  $Coupl_{EvOp}(ev_i, op_j) \in [0, 1]$  between an event  $ev_i$  of a semantically annotated Web API  $\mathcal{W}_i$  and an operation  $op_j$  of a semantically annotated Web API  $\mathcal{W}_j$  is evaluated as: 459  
460  
461  
462

$$Coupl_{EvOp}(ev_i, op_j) = \frac{\sum_{h,k} CAff(out_h, in_k)}{|OUT_{ev}(ev_i)|} \quad (11.2)$$

where  $OUT_{ev}(ev_i)$  is the set of concepts used to annotate the event outputs of the  $ev_i$  event,  $out_h \in OUT_{ev}(ev_i)$ ,  $in_k$  is a concept used to annotate the  $k$ th input of the  $op_j$  operation. 463  
464  
465

The  $Coupl_{EvOp}(ev_i, op_j)$  formula measures how many outputs of the  $ev_i$  event are caught by an input of the  $op_j$  operation. This formula is used to compute the *functional coupling degree* as follows. 466  
467  
468

**Definition 11.4 (Functional Coupling Degree).** The Functional Coupling Degree  $Coupl_{IO}(\mathcal{W}_i, \mathcal{W}_j) \in [0, 1]$  between two semantically annotated Web APIs  $\mathcal{W}_i$  and  $\mathcal{W}_j$  is evaluated as: 469  
470  
471

$$Coupl_{IO}(\mathcal{W}_i, \mathcal{W}_j) = \frac{\sum_{i,j} Coupl_{EvOp}(ev_i, op_j)}{|EV(\mathcal{W}_i)|} \quad (11.3)$$

where  $ev_i \in EV(\mathcal{W}_i)$  and  $op_j \in OP(\mathcal{W}_j)$ . 472

The  $Coupl_{IO}(\mathcal{W}_i, \mathcal{W}_j)$  formula measures how much the events raised on the  $\mathcal{W}_i$  Web API interface can be coupled with operations provided by the  $\mathcal{W}_j$  Web API. When the provider publishes a new semantically annotated Web API description, a preliminary search of the available Web APIs on the Web of Linked Data is performed to find those Web APIs whose operation inputs are annotated with concepts related to the ones used to annotate the event outputs of the Web API to be published. Among these Web APIs, the ones which present a functional coupling degree value equal to or greater than the threshold  $\theta$  are proposed to the provider to set the `potentialCouplingWith` link between them.

The main task to be performed by a Web designer when he/she is creating and linking a new Web mashup is to find relevant Web APIs on the Web of Linked Data. To this purpose, he/she exploits the network of `similarTo` and `potentialCouplingWith` links, as we will show in the next section. After selecting the Web APIs to be included in the new Web mashup, the Web designer must provide activation relationships between pairs of coupled APIs. To this purpose, the Web designer is supported for the identification of IO correspondences between event outputs and operation inputs. Also in this case, the `CAff()` evaluation is used to suggest potential couplings. Finally, IO correspondence is manually confirmed by the Web designer and relies on the lowering and lifting schema mappings provided together with the semantically annotated Web APIs (see Fig. 11.2).

## 11.6 Selection Strategies for Linked Web API Mashup

493

We distinguish two Web mashup development scenarios that could take benefits from the organization of semantically annotated Web APIs on top of the Web of Linked Data. We analyze the scenarios in the following and then we formalize a solution to support the Web designer in completing his/her task.

*Building a mashup from the scratch.* Let us consider Veruschka who needs to build from scratch the mashup application described in Sect. 11.2 and to link it to the Web of Linked Data. The designer proceeds step by step by selecting a Web API and setting activation relationships between events and operations, according to the model presented in the Sect. 11.4. The designer specifies a request for a Web API in terms of desired categories, operations, and/or I/Os. A ranked list of available Web APIs should be proposed. Once the designer selected one of the proposed APIs from the list, say  $\mathcal{W}_s$ , and placed it in the Web mashup under development, two additional lists of Web APIs can be suggested: (1) APIs that are similar to the selected one, that is, are related to  $\mathcal{W}_s$  through a `similarTo` link (Veruschka can use one of them in the Web mashup as an alternative to  $\mathcal{W}_s$ ), and (2) APIs that can be coupled with the selected one to implement additional functionalities in the mashup, that is, are related to  $\mathcal{W}_s$  through a `potentialCouplingWith` link (e.g., Veruschka, after selecting a Web API that provides the addresses of movie theaters, could add a GoogleMaps API to locate addresses on a map). For each additional coupled API, IO correspondences among inputs of its operations and outputs of  $\mathcal{W}_s$  events are

suggested on demand to the designer to model activation relationships. Exploration 514  
of these suggestions allows the designer to gain better insight into the interpretation 515  
of the Web API descriptions within the Web mashup under development. 516

*Modifying an existing application.* Let us now consider Veruschka who needs to 517  
modify the Web mashup by replacing one or more descriptors. Similarly, we could 518  
consider another Web designer, Caecilius, who finds Veruschka's Web mashup on 519  
the Web of Linked Data and desires to replace one of the component Web APIs. 520  
Web APIs can be substituted because of (1) they become unavailable, (2) application 521  
requirements have been changed, and (3) the designer needs to improve the QoS of 522  
the Web mashup [12]. Once the designer selects a Web API to be substituted, say 523  
 $\mathcal{W}_s$ , a ranked list of alternative APIs are suggested as candidates for replacing  $\mathcal{W}_s$ . In 524  
this case, a good alternative to  $\mathcal{W}_s$  is an API that is highly similar to  $\mathcal{W}_s$  and highly 525  
coupled with other APIs which were connected to  $\mathcal{W}_s$  in the original mashup. The 526  
designer compares the features of candidate descriptors to decide the most suitable 527  
alternatives. 528

The analysis of the above development scenarios raised the need of implementing 529  
different kinds of *selection patterns* that rely on the semantic organization of Web 530  
APIs and mashups built upon the Web of Linked Data. In the following, we better 531  
formalize the patterns and we present the I-MASH platform which implements them. 532

### 11.6.1 Linked Web API Selection Patterns

533

Formally, a *selection pattern* is defined as follows. 534

**Definition 11.5 (Selection Pattern).** A selection pattern  $\sigma$  is a quadruple: 535

$$\sigma_\tau = \langle \mathcal{W}_\tau, m_\tau, \delta_\tau, \prec_\tau \rangle \quad (11.4)$$

where  $\tau$  is the goal of the selection pattern. If  $\tau = \text{'search'}$ , the aim 536  
is to suggest a set of Web APIs that match a Web API specification  $\mathcal{W}_\tau$ . 537  
If  $\tau = \text{'completion'}$ , or  $\tau = \text{'substitution'}$ , then the aim is to 538  
suggest a list of Web APIs that can be coupled or substituted to the Web API 539  
 $\mathcal{W}_\tau$ , respectively. The metric  $m_\tau$  is used to evaluate the degree of matching 540  
(if  $\tau = \text{'search'}$ ), the degree of coupling (if  $\tau = \text{'completion'}$ ) or 541  
the degree of similarity (if  $\tau = \text{'substitution'}$ ) between each suggested 542  
Web API and  $\mathcal{W}_\tau$ . The threshold  $\delta_\tau$  is used to filter out not relevant Web 543  
APIs. A Web API  $\mathcal{W}_j$  is proposed to the designer if  $m_\tau(\mathcal{W}_\tau, \mathcal{W}_j) > \delta_\tau$ . 544  
Finally,  $\prec_\tau$  is a ranking function to present the suggested Web APIs. In 545  
particular,  $\mathcal{W}_i \prec_\tau \mathcal{W}_j$ , that is,  $\mathcal{W}_i$  is ranked higher than  $\mathcal{W}_j$ , if  $m_\tau(\mathcal{W}_\tau, \mathcal{W}_i) \geq 546$   
 $m_\tau(\mathcal{W}_\tau, \mathcal{W}_j)$ . 547

We distinguish two kinds of selection patterns: (1) a selection pattern that 548  
requires the formulation of a query on the Web of Linked APIs, as formulated 549  
by Veruschka, to find Web APIs that satisfy some search criteria (*search* selection 550

pattern) and (2) selection patterns which start from the selection of a Web API 551 previously included in the Web mashup under development (*completion* and 552 *substitution* selection patterns). In the latter case, the selected Web API is one of 553 the APIs available in the Web of Linked Data, and the net of *similarTo* and 554 *potentialCouplingWith* links is browsed to evaluate alternative APIs or Web 555 APIs that can be added to the mashup. In the former case, the requested Web API is 556 defined as the following query: 557

$$\mathcal{W}_r = \langle \text{CAT}(\mathcal{W}_r), \text{opt}(\mathcal{W}_r) \rangle \quad (11.5)$$

*CAT*( $\mathcal{W}_r$ ) are the required categories, and  $\text{opt}(\mathcal{W}_r) = \langle \text{OP}(\mathcal{W}_r), \text{IN}(\mathcal{W}_r), \text{OUT}(\mathcal{W}_r) \rangle$  represent the optional sets of required operation names, input names, 558 and output names. With respect to the Web API model in Sect. 11.4, the query 559 560  $\mathcal{W}_r$  has a flattened structure, since the sets  $\text{IN}(\mathcal{W}_r)$  and  $\text{OUT}(\mathcal{W}_r)$  are specified 561 independently from the operation in  $\text{OP}(\mathcal{W}_r)$  they belong to. In fact, according 562 to an exploratory search perspective, Veruschka does not have a very precise idea 563 about the structure of the Web APIs to search. Veruschka specifies the categories 564 and, optionally, the inputs, the outputs, and the functionalities (i.e., the operations) 565 expected from the API. 566

Metrics are based on the *similarTo* and *potentialCouplingWith* links 567 between Web API URIs on the Web of Linked Data and on the evaluation of 568 similarity and coupling degree. 569

Let  $\tau = \text{'completion,'}$   $\mathcal{W}_c$  a selected Web API already included in the Web 570 mashup under development (identified through a URI  $\text{Wc\_uri}$ ) and  $\text{cat}_c$  a desired 571 category. The goal is now to find Web APIs  $\mathcal{W}_j$  classified in the  $\text{cat}_c$  category that 572 can be coupled to  $\mathcal{W}_c = \mathcal{W}_\tau$ . The metric  $m_\tau$  used to suggest candidate Web APIs 573  $\mathcal{W}_j$  is defined as 574

$$m_\tau = \begin{cases} \text{Coup}_{\text{IO}}(\mathcal{W}_c, \mathcal{W}_j) & \text{if } \mathcal{W}_j \in Q^1(\mathcal{W}_j) \\ 0 & \text{otherwise} \end{cases} \quad (11.6)$$

where  $Q^1(\mathcal{W}_j)$  is a SPARQL query that retrieves those Web APIs that have a 575 category equivalent to or less specific than  $\text{cat}_c$  and are related to  $\mathcal{W}_c$  through a 576 *potentialCouplingWith* link. 577

```
PREFIX lwa: ...
...
SELECT ?uri
WHERE {
  ?api lwa:hasURI ?uri ;
    lwa:hasCategory ?cat .
  Wc_uri lwa:potentialCouplingWith ?api .
  {
    {?cat owl:equivalentTo cat_c .} UNION
    {cat_c rdfs:subClassOf ?cat .}
  }
}
```

Similarly, let  $\tau = \text{'substitution'}$ ,  $\mathcal{W}_s$  a selected Web API already included in the Web mashup under development (identified through a URI  $\text{Ws\_uri}$ ),  $\{\mathcal{W}_k\}$  a set of Web APIs in the Web mashup (identified through URIs  $\{\text{Wk\_uri}\}$ ) such that there exists an activation relationship between  $\mathcal{W}_s$  and each  $\mathcal{W}_k$ , and  $\{\mathcal{W}_h\}$  a set of Web APIs in the mashup (identified through URIs  $\{\text{Wh\_uri}\}$ ) such that there exists an activation relationship between each  $\mathcal{W}_h$  and  $\mathcal{W}_s$ . The goal is now to find Web APIs  $\mathcal{W}_j$  that substitute  $\mathcal{W}_s$  and can be coupled with  $\{\mathcal{W}_k\}$  and  $\{\mathcal{W}_h\}$ . The function  $m_\tau$  is made up of three parts:

- The functional similarity degree between  $\mathcal{W}_s$  and  $\mathcal{W}_j$ , that is,  $\text{Sim}_{\text{IO}}(\mathcal{W}_s, \mathcal{W}_j)$
- The sum of the functional coupling degree between  $\mathcal{W}_j$  and each  $\mathcal{W}_k$
- The sum of the functional coupling degree between  $\mathcal{W}_h$  and  $\mathcal{W}_j$

That is,

589

$$m_\tau = \begin{cases} \text{Sim}_{\text{IO}}(\mathcal{W}_s, \mathcal{W}_j) + \sum_k \text{Coup}_{\text{IO}}(\mathcal{W}_j, \mathcal{W}_k) + \sum_h \text{Coup}_{\text{IO}}(\mathcal{W}_h, \mathcal{W}_j) & \text{if } \mathcal{W}_j \in Q^2(\mathcal{W}_j) \\ 0 & \text{otherwise} \end{cases} \quad (11.7)$$

where  $Q^2(\mathcal{W}_j)$  is a SPARQL query that retrieves those Web APIs that have a category equivalent to or less specific than at least a category of  $\mathcal{W}_s$ , is related to  $\mathcal{W}_s$  through a `similarTo` link, and is related to each  $\mathcal{W}_k$  and each  $\mathcal{W}_h$  through a `potentialCouplingWith` link.

```
PREFIX lwa: ...
...
SELECT ?uri
WHERE {
  ?api lwa:hasURI ?uri ;
    lwa:hasCategory ?cat ;
    lwa:similarTo Ws_uri ;
    lwa:potentialCouplingWith ?Wk_uri ; ## for each Wk_uri
  ...
  Wh_uri lwa:potentialCouplingWith ?api . ## for each Wh_uri
  ...
  Ws_uri lwa:hasCategory ?cat_s .
  {
    {?cat owl:equivalentTo ?cat_s .} UNION
    {?cat_s rdfs:subClassOf ?cat .}
  }
}
```

Finally, let be  $\tau = \text{'search'}$  and let be  $\mathcal{W}_r$  the requested Web API (i.e.,  $\mathcal{W}_\tau = \mathcal{W}_r$ ). The metric  $m_\tau$  used to suggest candidate Web APIs  $\mathcal{W}_j$  that match the request  $\mathcal{W}_r$  is defined as:

596

$$m_\tau = \begin{cases} \text{Sim}_{\text{IO}}(\mathcal{W}_r, \mathcal{W}_j) & \text{if } \mathcal{W}_j \in Q^3(\mathcal{W}_j) \text{ AND } \text{opt}(\mathcal{W}_r) \neq \langle \emptyset, \emptyset, \emptyset \rangle \\ 1 & \text{if } \mathcal{W}_j \in Q^3(\mathcal{W}_j) \text{ AND } \text{opt}(\mathcal{W}_r) = \langle \emptyset, \emptyset, \emptyset \rangle \\ 0 & \text{otherwise} \end{cases} \quad (11.8)$$

597

In this  $m_\tau$  evaluation of the  $\text{Sim}_{\text{IO}}()$  expression (see (11.1)), the terms corresponding to optional parts of the request are set to 0.  $Q^3(\mathcal{W}_j)$  is a SPARQL query that retrieves those Web APIs that have a category equivalent to or less specific than at least a category of  $\mathcal{W}_r$ . 598  
599  
600  
601

```
PREFIX lwa: ...
...
SELECT ?uri
WHERE {
  ?api lwa:hasURI ?uri ;
    lwa:hasCategory ?cat .
  {
    {?cat owl:equivalentTo cat_r .} UNION
    {cat_r rdfs:subClassOf ?cat .}
  }
}
```

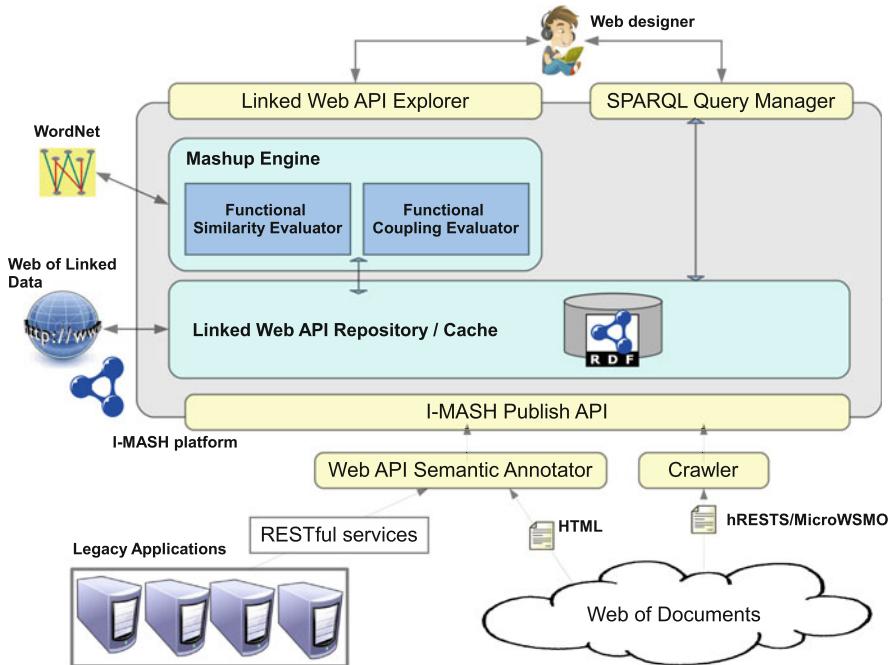
## 11.7 The I-MASH Platform

602

Figure 11.3 shows the architecture of the I-MASH platform, which implements the 603 Linked Web APIs publication and selection procedures. In the following, we present 604 the main modules which compose the platform. 605

*Mashup Engine.* This is the core module of the I-MASH platform. The *Mashup* 606 *Engine* implements the selection patterns detailed in Sect. 11.6 and includes 607 the modules for functional similarity and functional coupling evaluation. The 608 *Mashup Engine* also presents a series of methods for the evaluation of the 609 *CAff* coefficient (see Sect. 11.5). To this aim, the *Mashup Engine* relies on 610 the terminological knowledge provided by WordNet lexical system [23] and on 611 the domain-specific knowledge from the Web of Linked Data that is accessed 612 through the *Linked Web API Repository*. 613

*Functional Similarity Evaluator.* Computes the functional similarity degree 614 according to the Definition 11.2. To this aim, this module relies on the methods 615 for *CAff* coefficient computation provided in the *Mashup Engine*. Similarly, 616 the *functional coupling evaluator* computes the functional coupling degree 617 according to the Definitions 11.3 and 11.4. 618



**Fig. 11.3** The I-MASH platform architecture

*Linked Web API Repository.* Maintains the representation of Linked Web APIs 619 published according to the model presented in Sect. 11.4. According to that 620 model, the Linked Web API descriptions refer to things (i.e., URIs of ontological 621 concepts) taken from the Web of Linked Data, which the *Linked Web API* 622 Repository is built upon. Other I-MASH modules, such as the *Mashup Engine* 623 and the *I-MASH Publish API*, exploit the *Linked Web API Repository* to access 624 the Web of Linked Data. 625

*I-MASH Publish API.* Implements the publication procedures described in 626 Sect. 11.5. To this aim, this module interacts with all the other modules of 627 the I-MASH platform. The *I-MASH Publish API* module is the entry point for 628 the external modules, such as crawlers or Web API semantic annotators, to 629 populate the *Linked Web API Repository* with semantically annotated Web APIs 630 extracted from RESTful services (e.g., used to access legacy application [41]) 631 or plain HTML documents used to access the wealth of data on the Web which 632 remain largely unexploited. Currently, we are developing an extension of the 633 SWEET tool [35] for taking into account also Web API events, thus enabling UI 634 integration as explained in Sect. 11.4. 635

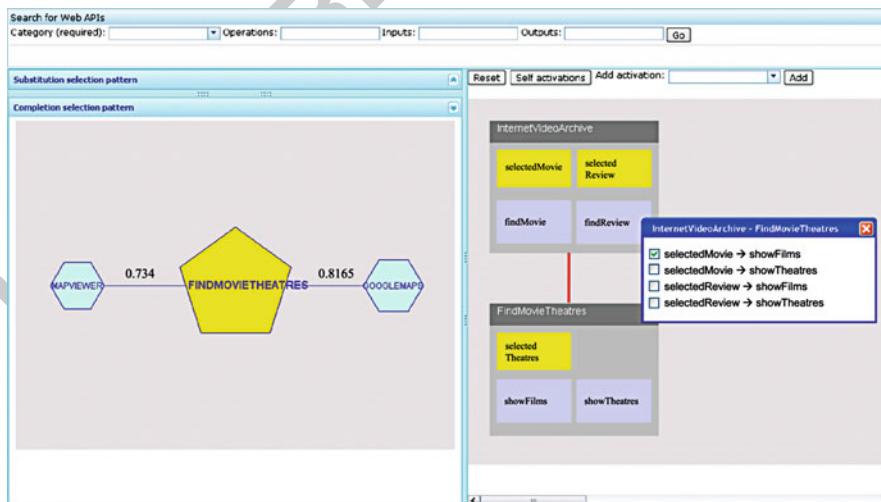
*Linked Web API Explorer.* This module supports the Web designer to find Web API 636 descriptions and compose them in a Web mashup. The module relies on the Web 637

API selection patterns implemented in the <i>Mashup Engine</i> and on the net of similarTo and potentialCouplingWith links in the <i>Linked Web API Repository</i> . The explorer is implemented as a Web-based interface and will be shown in the next section.	638 639 640 641
<i>SPARQL Query Manager</i> . For skilled Web designers, we equipped the I-MASH platform with an interface to directly query the <i>Linked Web API Repository</i> through the formulation and execution of SPARQL queries.	642 643 644

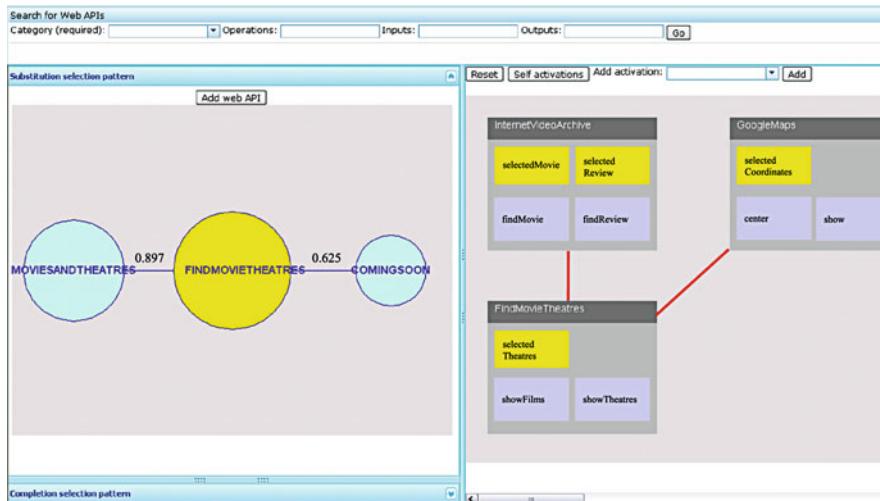
### 11.7.1 Exploring Linked Web APIs

645

We describe a preliminary, proof-of-concept implementation of the Linked Web API Explorer, showing its application within the scenarios introduced at the beginning of Sect. 11.6. The Linked Web API Explorer has been implemented as a Web-based GUI using the ZK open framework, which includes a library of AJAX components. Two screenshots of the explorer are shown in Figs. 11.4 and 11.5. The screenshots are based on the motivation example and show the exploration of the Linked Web API Repository through the application of the completion and substitution selection patterns. In the upper part of the Web-based GUI, the Web designer can perform the search for available Web APIs by specifying their category (Search for Web APIs fields) and, optionally, operation, input, and output names. On the right, the explorer provides a canvas where the Web mashup can be incrementally built. The Web designer can drag-and-drop component Web APIs from the panels on the left, 655



**Fig. 11.4** Linked Web API Explorer: application of the completion selection pattern



**Fig. 11.5** Linked Web API Explorer: application of the substitution selection pattern

where APIs are shown following `similarTo` and `potentialCouplingWith` links in the *Linked Web API Repository*.  
658  
659

Let us assume that Veruschka has already selected the InternetVideo  
660 Archive and the FindMovieTheatres Web APIs and put them on the  
661 canvas as shown in Fig. 11.4. Lines connecting the rectangles in the canvas  
662 represent event-operation activations between Web APIs in the Web  
663 mashup. In this example, when a movie is selected on the InternetVideoArchive  
664 API (`selectedMovie` event), theaters showing the selected movie are  
665 displayed on the FindMovieTheatres API (`showFilms` operation). By  
666 pushing the “Self activations” button, the explorer suggests IO  
667 correspondences and event-operation mappings between Web APIs in the canvas.  
668 The mappings between events and operations are shown by double-clicking  
669 the activation relationship between the APIs. According to the completion  
670 selection pattern, if Veruschka selects the FindMovieTheatres API in the  
671 canvas, the explorer suggests him the GoogleMaps and MapViewer APIs,  
672 where the  $m_{completion}(FindMovieTheatres, GoogleMaps) = 0.8165$  and  
673  $m_{completion}(FindMovieTheatres, MapViewer) = 0.734$ . The suggested  
674 Web APIs are shown on the left within the “completion selection pattern” panel.  
675 The selected API `FindMovieTheatres` is highlighted as a pentagon in the  
676 center of the panel, while the suggested Web APIs are shown as hexagons around  
677 the pentagon; the size of each hexagon is proportional to the  $m_{completion}$  values.  
678

Let us now assume that Veruschka has added the GoogleMaps API in the  
679 canvas and aims at substituting the FindMovieTheatres API in the Web appli-  
680 cation under development (see Fig. 11.5). According to the substitution selection  
681

pattern, the explorer suggests to Veruschka the MoviesAndTheatres and ComingSoon APIs, where  $m_{\text{substitution}}(\text{FindMovieTheatres}, \text{Coming Soon}) = 0.625$  and  $m_{\text{substitution}}(\text{FindMovieTheatres}, \text{MoviesAndTheatres}) = 0.897$ . The selected API FindMovieTheatres is highlighted as a circle in the center of the “Substitution selection pattern” panel, while the suggested APIs are displayed as circles around FindMovieTheatres. The size of each circle is proportional to the  $m_{\text{substitution}}$  values.

## 11.8 Conclusion

Recently, Web APIs have been more and more used to access documents and metadata from the Web of Linked Data. When the selection from the shelf of available Web APIs and their composition are performed to build value-added, short-living, situational applications, the term *mashup* is often used [5]. Different lightweight solutions have been proposed for the semantic annotation of Web APIs, like the combined use of MicroWSMO/hRESTS languages [32]. However, given the growing number of available APIs, users are overwhelmed by the quantity of data that are available to be composed. Moreover, semantic mashup of Linked Data (services) often assumes an information retrieval perspective, where the mashup designer has a specific query in mind which he/she tries to answer. Finally, mashup of Web APIs often involves the integration of UIs (also called widgets or gadgets) that is often event driven [14]. In this chapter, we addressed the above-mentioned issues by proposing a new approach for “serving up” Linked Data by semantics-enabled mashup of existing Web APIs. A basic semantics-enabled model of Web APIs is proposed. The model has been designed (a) to support providers who publish new Web APIs used to access the Web of Linked Data, (b) to support the Web designer who aims at exploring and selecting available Web APIs to build or maintain a Web mashup, and (c) to make it available on top of the Web of Linked Data. Based on the proposed model, we describe automated matching techniques apt to establish *semantic links* among Linked Web APIs according to properly defined criteria: (1) *functional similarity links*, set between APIs that provide similar functionalities and enable access to similar Linked Data, and (2) *functional coupling links*, set between APIs that can be wired to realize new, value-added Web mashups. Finally, we identified recurrent use cases to support the construction of Web mashups by means of Linked Web API composition. The analyzed use cases enable a proactive suggestion and interactive selection of available Linked Web APIs in order to support the browsing of available resources according to an exploratory perspective, that is, a step-by-step construction of the final mashup, which evolves depending on the available APIs and the associated links. The model and the techniques we propose adhere to the Linked Data principles for publishing resources on the Web in a meaningful way: (1) HTTP URIs are used to identify published resources (i.e., Web APIs), (2) useful (RDF) information are provided on the Web API description when someone looks up a Web API through its URI,

and (3) functional similarity links and functional coupling links are set to relate 723  
 Web APIs to each other, thus enabling easy development and sharing of new Web 724  
 applications. Moreover, we rely on the Web of Linked Data for Web API semantic 725  
 annotation, where Linked Data are used as source of knowledge. A preliminary, 726  
 proof-of-concept implementation of the Linked Web API Explorer has also been 727  
 presented. 728

An extensive experimentation of the proposed platform in several real-case 729  
 application scenarios is being studied. Moreover, an extension of the proposed 730  
 model to include also nonfunctional aspects (see, e.g., the application of quality 731  
 issues to the mashup of components [12]) will be studied. Finally, the proposed 732  
 selection patterns will be enriched to consider QoS aspects, context-aware Web 733  
 mashup development, and complementary perspectives, like the one presented 734  
 in [26]. 735

## References

736

- AQ2
1. W3C SWEO Community, Linking Open Data project. <Http://esw.w3.org/topic/SweoIG/> 737  
[TaskForces/CommunityProjects/LinkingOpenData](#) 738
  2. Abiteboul, S., Greenshpan, O., Milo, T.: Modeling the mashup space. Proceedings of the 739  
 workshop on web information and data management, pp. 87–94 (2008) 740
  3. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., 741  
 Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business process execution 742  
 language for web services version 1.1 (2003). <http://www-128.ibm.com/developerworks/library/specification/wsbpel/> 743  
 744
  4. Becker, C., Bizer, C.: DBpedia mobile: a location-enabled linked data browser. Proceedings of 745  
 WWW the International Workshop on Linked Data on the Web (LDOW08) (2008) 746
  5. Benslimane, D., Dustdar, S., Sheth, A.: Service mashups: the new generation of web 747  
 applications. IEEE Inter. Comput. **12**(5), 13–15 (2008) 748
  6. Berners-Lee, T., Hendler, J.A., Lassila, O.: The semantic web. Sci. Am. **284**(5), 34–43 (2001) 749
  7. Berners-Lee, T., Hollenbach, J., Lu, K., Presbrey, J., d'ommeaux, E.P., Schraefel, M.: Tabulator 750  
 redux: writing into the semantic web. Technical report, ECSIAMEprint14773, Electronics and 751  
 Computer Science, University of Southampton (2007) 752
  8. Bianchini, D., Antonellis, V.D., Melchiori, M.: Flexible semantic-based service matchmaking 753  
 and discovery. World Wide Web J. **11**(2), 227–251 (2008) 754
  9. Bianchini, D., Antonellis, V.D., Melchiori, M.: Semantic-driven mashup design. Proceedings 755  
 of the 12th International Conference on Information Integration and Web-based Applications 756  
 and Services (iiWAS'10), pp. 245–252 (2010) 757
  10. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. Int. J. Semant. Web Inform. 758  
 Syst. **5**(3), 1–22 (2009) 759
  11. Bussler, C., de Bruijn, J., Feier, C., Fensel, D., Keller, U., Lara, R., Lausen, H., Polleres, 760  
 A., Roman, D., Stollberg, M.: Web service modeling ontology. Appl. Ontol. **1**(1), 77–106 761  
 (2005) IOS Press 762
  12. Cappiello, C., Daniel, F., Matera, M., Pautasso, C.: Information quality in mashups. Inter. 763  
 Comput. **14**(4), 14–22 (2010) 764
  13. Chumbley, R., Durand, J., Pilz, F., Rutt, T.: WS-interoperability basic profile version 2.0. 765  
 Technical report, Organization for the Advancement of Structured Information Standards 766  
 (OASIS) (2010). URL <http://www.ws-i.org/> 767

14. Daniel, F., Casati, F., Benatallah, B., Shan, M.: Hosted universal composition: models, languages and infrastructure in mashart. Proceedings of the 28th International Conference on Conceptual Modeling (ER'09) (2009)
15. d'Aquin, M., Motta, E., Sabou, M., Angeletou, S., Gridinoc, L., Lopez, V., Guidi, D.: Toward a new generation of semantic web applications. IEEE Intel. **23**(3), 20–28 (2008)
16. Davies, J., Domingue, J., Pedrinaci, C., Fensel, D., Gonzalez-Cabero, R., Potter, M., Richardson, M., Stinicic, S.: Towards the open service web. British Telecommun. Technol. J. **26**(2) (2009)
17. De Virgilio, R., Orsi, G., Tanca, L., Torlone, R.: Reasoning over large semantic datasets. Proceedings of the 17th Italian Symposium on Advanced Database Systems. Camogli (Genova), Italy (2009)
18. Edwards, D., Hardman, L.: Hypertext: theory into practice, chap. Lost in Hyperspace: cognitive mapping and navigation in a hypertext environment, pp. 90–105. Exeter, UK (1999)
19. Elmeleegy, H., Ivan, A., Akkiraju, R., Goodwin, R.: MashupAdvisor: a recommendation tool for mashup development. Proceedings of the 6th International Conference on Web Services (ICWS'08), pp. 337–344. Beijin, China (2008)
20. Ennals, R., Garofalakis, M.: MashMaker: mashups for the masses. Proceedings of the 27th ACM SIGMOD International Conference on Management of Data, pp. 1116–1118 (2007)
21. Erl, T.: SOA Principles of Service Design. Prentice Hall, Englewood, Cliffs, NJ (2007)
22. Farrell, J., Lausen, H.: Semantic annotations for WSDL and XML schema. Recommendation, W3C (2007)
23. Fellbaum, C.: Wordnet: An Electronic Lexical Database. MIT, Cambridge, MA (1998)
24. Fielding, R.: Architectural styles and the design of network-based software architectures. Ph.D. Thesis, University of California, Irvine (2000)
25. Gomadam, K., Ranabahu, A., Nagarajan, M., Sheth, A.P., Verma, K.: A faceted classification based approach to search and rank web APIs. ICWS, pp. 177–184 (2008)
26. Greenshpan, O., Milo, T., Polyzotis, N.: Autocompletion for mashups. Proceedings of the 35th International Conference on very Large Databases (VLDB'09), pp. 538–549. Lyon, France (2009)
27. Hadley, M.: Web application description language. Technical report, W3C (2009)
28. Hately, L., von Riegen, C., Rogers, T.: UDDI specification version 3.0.2. Technical report, organization for the advancement of structured information standards (OASIS) (2004)
29. Hepp, M.: Products and services ontologies: a methodology for deriving owl ontologies from industrial categorization standards. Int. J. Semant. Web Inform. Syst. **2**(1), 72–99 (2006)
30. Jarar, M., Dikaiakos, M.: A data mashup language for the data web. Proceedings of the 2nd Workshop on Linked Data on the Web (LDOW09). Madrid, Spain (2009)
31. Kopecky, J.: WSDL RDF mapping: developing ontologies from standardized XML languages. Proceedings of the 1st International ER Workshop on Ontologizing Industrial Standards (OIS'06). Tucson, Arizona, USA (2006)
32. Kopecky, J., Vitvar, T., Fensel, D.: hRESTS & MicroWSMO. Tech. rep., SOA4ALL Project, Deliverable D3.4.3 (2009)
33. Lathem, J., Gomadam, K., Sheth, A.: SA-REST and (s)mashups: adding semantics to RESTful services. Proceedings of the IEEE International Conference on Semantic Computing, pp. 469–476. IEEE CS Press (2007)
34. Lu, B., Wu, Z., Ni, Y., Xie, G., Zhou, C., Chen, H.: sMash: semantic-based mashup navigation for data API network. Proceedings of the 18th International World Wide Web Conference, pp. 1133–1134 (2009)
35. Maleshкова, M., Pedrinaci, C., Domingue, J.: Semantic annotation of Web APIs with SWEET. Proceedings of the 6th Workshop on Scripting and Development for the Semantic Web (2010)
36. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: semantic markup for web services, v1.1. Technical report, World Wide Web Consortium (W3C) (2004)
37. Nadalin, A., Kaler, C., Monzillo, R., Hallam-Backer, P.: Web services security: SOAP message security 1.1. Technical report, Organization for the Advancement of Structured Information Standards (OASIS) (2006). URL <http://docs.oasis-open.org/wss/v1.1>

38. Ngu, A.H.H., Carlson, M.P., Sheng, Q.Z., Paik, H.: Semantic-based mashup of composite applications. *IEEE T. Serv. Comput.* **3**(1), 2–15 (2010) 823  
824
39. Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., Tummarello, G.: Sindice.com: a document-oriented lookup index for open linked data. *Int. J. Metadata Semant. Ontol.* **3**(1), 825  
826  
827  
37–52 (2008)
40. Pedrinaci, C., Domingue, J., Krummenacher, R.: Services and the web of data: an unexploited symbiosis. *Linked AI: AAAI spring symposium on linked data meets artificial intelligence* 828  
829  
(2010) 830
41. Pedrinaci, C., Liu, D., Maleshkova, M., Lambert, D., Kopecky, J., Domingue, J.: iServe: a 831  
linked services publishing platform. *Proceedings of ESWC Ontology Repositories and Editors* 832  
for the Semantic Web (2010) 833
42. Phuoc, D.L., Polleres, A., Morbidoni, C., Hauswirth, M., Tummarello, G.: Rapid prototyping 834  
of semantic mash-ups through semantic web pipes. *Proceedings of the 18th World Wide Web* 835  
Conference (WWW09), pp. 581–590. Madrid, Spain (2009) 836
43. Pilioura, T., Tsagatidou, A.: Unified publication and discovery of semantic web services. *ACM* 837  
*Trans. Web* **3**(3), 1–44 (2009) 838
44. Riabov, A., Boillet, E., Feblowitz, M., Liu, Z., Ranganathan, A.: Wishful search: interactive 839  
composition of data mashups. *Proceedings of the 19th International World Wide Web* 840  
Conference (WWW'08), pp. 775–784. Beijin, China (2008) 841
45. Richardson, L., Ruby, S.: RESTful web services. O'Reilly (2007) 842
46. van Rijsbergen, C.J.: Information Retrieval. Butterworth, London (1979) 843
47. Vitvar, T., Kopecky, J., Viskova, J., Fensel, D.: WSMO-lite annotations for Web Services. 844  
*Proceedings of the 5th European Semantic Web Conference* (2008) 845

AUTHOR QUERIES

- AQ1. Kindly check the corresponding author.
- AQ2. Kindly provide the accessed date for Ref. [1].

UNCORRECTED PROOF

**Part III<sup>1</sup>**  
**Linked Data Search Engines<sup>2</sup>**

UNCORRECTED PROOF

# Chapter 12

## A Recommender System for Linked Data

Roberto Mirizzi, Azzurra Ragone, Tommaso Di Noia,  
and Eugenio Di Sciascio

### 12.1 Introduction

Peter and Alice are at home, it is a calm winter night, snow is falling, and it is too cold to go outside. “*Why don’t we just order a pizza and watch a movie?*” says Alice wrapped in her favorite blanket. “*Why not?*”—Peter replies—“*Which movie do you wanna watch?*” “*Well, what about some comedy, romance-like one? Com’on Pete, look on Facebook, there is that nice application that Kara suggested me some days ago!*” answers Alice. “*Oh yes, MORE, here we go, tell me a movie you like a lot,*” says Peter excited. “*Uhm, I wanna see something like the Bridget Jones’s Diary or Four Weddings and a Funeral, humour, romance, good actors...*” replies his beloved, rubbing her hands. Peter is a bit concerned, he is more into fantasy genre, but he wants to please Alice, so he looks on MORE for movies similar to the Bridget Jones’s Diary and Four Weddings and a Funeral: “*Here we are my dear, MORE suggests the sequel or, if you prefer, Love Actually,*” I would prefer the second.” “*Great! Let’s rent it!*” nods Peter in agreement.

The scenario just presented highlights an interesting and useful feature of a modern Web application. There are tasks where the users look for items similar to the ones they already know. Hence, we need systems that recommend items based on user preferences. In other words, systems should allow an easy and friendly exploration of the information/data related to a particular domain of interest. Such characteristics are well known in the literature and in common applications such as recommender systems [23]. Nevertheless, new challenges in these fields arise when

---

R. Mirizzi (✉) · T. Di Noia · E. Di Sciascio

Politecnico di Bari, via Orabona 4 – 70125 Bari, Italy

e-mail: [mirizzi@deemail.poliba.it](mailto:mirizzi@deemail.poliba.it); [t.dinoia@poliba.it](mailto:t.dinoia@poliba.it); [disciascio@poliba.it](mailto:disciascio@poliba.it)

A. Ragone

Politecnico di Bari, via Orabona 4 – 70125 Bari, Italy

Exprivia S.p.A., viale A. Olivetti 11/A – 70056 Molfetta, BA, Italy

e-mail: [a.ragone@poliba.it](mailto:a.ragone@poliba.it); [azzurra.ragone@exprivia.it](mailto:azzurra.ragone@exprivia.it)

the information used by these systems exploits the huge amount of interlinked data 26  
coming from the semantic Web. 27

Thanks to the Linked Data initiative, the foundations of the semantic Web 28  
have been built [4]. Shared, open, and linked RDF datasets give us the possibility 29  
to exploit both the strong theoretical results and the robust technologies and tools 30  
developed since the semantic Web appearance in 2001 [3]. In a simplistic way, we 31  
may think at the semantic Web as a huge distributed database we can query to get 32  
information coming from different sources. Usually, datasets expose a SPARQL [22] 33  
endpoint to make the data accessible through exact queries. If we know the URI of 34  
the actress *Keira Knightley* in DBpedia<sup>1</sup> [5], we may retrieve all the movies she 35  
acted with a simple SPARQL query such as the following: 36

```
SELECT ?movie WHERE {
  ?movie <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://dbpedia.org/ontology/Film> .
  ?movie <http://dbpedia.org/ontology/starring>
    <http://dbpedia.org/resource/Keira_Knightley>
}
```

Although these are very exciting results and applications, there is much more 43  
behind the curtains. Datasets come with the description of their schema structured 44  
in an ontological way. Resources refer to classes which are in turn organized in 45  
well-structured and rich ontologies. Exploiting this feature further, we go beyond 46  
the notion of a distributed database and we can refer to the semantic Web as a 47  
distributed social knowledge base. Its social characteristic is inherited by the social 48  
nature of the information sources they are extracted from. This is a strong point in 49  
favor of such semantic datasets since there is no need to manually maintain and 50  
update the knowledge bases themselves, as they are automatically extracted from 51  
user-contributed content. As an example, consider DBpedia which is the RDF 52  
version of Wikipedia. 53

In this chapter, we present MORE, a system for *movie recommendation* in the 54  
Web of Data. Currently, the system relies on one of the most relevant datasets 55  
in the Linked Data project, DBpedia, and on the semantic-enabled version 56  
of the *Internet Movie Database* (IMDB): LinkedMDB<sup>2</sup> [13]. Hence, the movie 57  
exploration and recommendation exploit semantic information embedded in social 58  
knowledge bases (see Sect. 12.2) belonging to the Linked Data cloud. 59

Main contributions of this chapter are: 60

- A semantic-based vector space model for recommendation of items in Linked 61  
Data datasets 62
- Presentation of a Facebook application for movie recommendation exploiting 63  
semantic datasets 64
- Evaluation and validation of the approach with real users 65

<sup>1</sup>[http://dbpedia.org/resource/Keira\\_Knightley](http://dbpedia.org/resource/Keira_Knightley)

<sup>2</sup><http://www.imdb.com>, <http://www.linkedmdb.org>

The remainder of the chapter is structured as follows: in Sect. 12.2, we illustrate how we exploit semantic information contained in RDF datasets to compute semantic similarities between movies, then in Sect. 12.3, we describe the interface and possible uses of MORE. In Sect. 12.4, we show how to compute similarities between movies using a semantic-adaptation of the vector space model (VSM). Section 12.5 shows the results of our evaluation, and Sect. 12.6 reviews relevant related work. Conclusion closes the chapter.

## 12.2 Social Knowledge Bases for Similarity Detection

MORE exploits information coming from different knowledge bases in the Linked Data cloud to provide a recommendation based on the semantic similarities between movie descriptions. The recommendation may also rely on user preferences to recommend movies. Since MORE has been implemented as a Facebook application, in order to avoid the *cold start* problem typical of content-based recommender systems, when the user starts using it, we may retrieve information about the movies she likes by grabbing them from her Facebook profile.

We exploit semantic information contained in the RDF datasets to compute a semantic similarity between movies the user could like. To this extent, DBpedia can be viewed as a social knowledge base since it harvests automatically information provided by the community in Wikipedia. The DBpedia knowledge base has several advantages over existing knowledge bases: it is multidomain, it is based on a community agreement, it follows the Wikipedia changes (it is always updated), and it is multilingual. Thanks to its SPARQL endpoint, it is possible to ask sophisticated queries against the knowledge base, achieving a complexity that would not be possible just crawling Wikipedia. The current DBpedia release (3.6) contains about 54,000 movies; 53,000 actors; 18,000 directors; and 12,000 categories directly related to movies, just to cite a few. All these resources are related to each other by several ontology properties belonging to the DBpedia ontology. Each property has its own semantics, domain, and range. For instance, the ontology object property dbpedia-owl:starring has domain dbpedia-owl:Work and range dbpedia-owl:Person (see Sect. 12.5.1 for further information about movie-related dataset extraction).

The main benefit of using the ontology is that the corresponding data are clean and well structured. It is possible to ask complex queries to the DBpedia SPARQL endpoint<sup>3</sup> with high precision in the results. For example, suppose we were interested in knowing which are the movies where *Hugh Grant* and *Colin Firth* starred together, we could ask DBpedia the following SPARQL query:

<sup>3</sup><http://dbpedia.org/sparql>

```

SELECT ?movie WHERE {
  ?movie <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://dbpedia.org/ontology/Film> .
  ?movie <http://dbpedia.org/ontology/starring>
    <http://dbpedia.org/resource/Hugh_Grant> .
  ?movie <http://dbpedia.org/ontology/starring>
    <http://dbpedia.org/resource/Colin_Firth>
}

```

102  
103  
104  
105  
106  
107  
108  
109

to obtain the following result set:

```

http://dbpedia.org/resource/Bridget_Jones%27s_Diary_%28film%29
http://dbpedia.org/resource/Bridget_Jones:_The_Edge_of_Reason_%28film%29
http://dbpedia.org/resource/Love_Actually

```

111  
112  
113

Intuitively, it is logical to suppose that, e.g., *Bridget Jones's Diary*<sup>4</sup> and *Love Actually*<sup>5</sup> are somehow similar to each other, since they share part of the cast.  
However, this is not the unique similarity between the two movies. In order to obtain all the information shared between the two movies, we may pose the following SPARQL query:

```

SELECT * WHERE {
  <http://dbpedia.org/resource/Bridget_Jones%27s_Diary_%28film%29> ?p ?o .
  <http://dbpedia.org/resource/Love_Actually> ?p ?o
}

```

119  
120  
121  
122

If we wanted to find the movies that are most related to *Bridget Jones's Diary* using DBpedia, probably one of the very first places should be occupied just by *Love Actually*. In fact, the two movies share part of the cast as seen before, have the same producer, the same writer, and some categories in common (e.g., *2000s romantic comedy films* and *Films set in London*). Roughly speaking, the more features two movies have in common, the more they are similar. In a few words, a similarity between two movies (or two resources in general) can be detected if in the RDF graph:

- They are directly related: this happens if a movie is the sequel of another movie, as, for example, with *Bridget Jones: The Edge of Reason*<sup>6</sup> and *Bridget Jones's Diary*. In DBpedia, this state is handled by the ontological properties dbpedia-owl:subsequentWork and dbpedia-owl:previousWork (see Fig. 12.1a).
- They are the subject of two RDF triples having the same property and the same object (see Fig. 12.1b). Referring to the previous example, *Bridget Jones's Diary* and *Love Actually* have two actors in common. In the movie domain, we take into account about 20 properties, such as dbpedia-owl:starring, dbpedia-owl:director, dbpedia-owl:producer, dbpedia-owl:musicComposer. They have been automatically extracted via SPARQL

<sup>4</sup>[http://en.wikipedia.org/wiki/Bridget\\_Jones's\\_Diary\\_\(film\)](http://en.wikipedia.org/wiki/Bridget_Jones's_Diary_(film))

<sup>5</sup>[http://en.wikipedia.org/wiki/Love\\_Actually](http://en.wikipedia.org/wiki/Love_Actually)

<sup>6</sup>[http://en.wikipedia.org/wiki/Bridget\\_Jones:\\_The\\_Edge\\_of\\_Reason\\_\(film\)](http://en.wikipedia.org/wiki/Bridget_Jones:_The_Edge_of_Reason_(film))

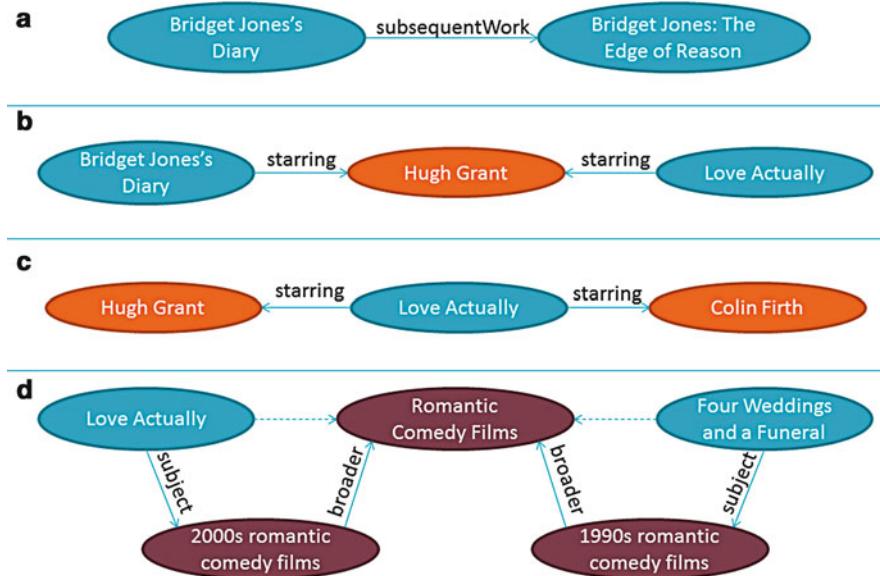


Fig. 12.1 The four types of graph matching involved in the similarity detection process

queries (see Sect. 12.5.1). The property `dc:terms :subject` needs a dedicated 142 discussion, as we will see in the following. 143

- They are the object of two RDF triples having the same property and the 144 same subject. This usually does not happen for movies in DBpedia. Anyway, 145 supposing we were interested in finding the similarities between actors, referring 146 to the previous example, we could find that *Hugh Grant* and *Colin Firth* are 147 related, since they acted in several movies together, i.e., they are the object of 148 two triples where the property is `dbpedia-owl :starring` and the subject 149 is, e.g., `dbpedia-res :Love_Actually` (Fig. 12.1c). 150

### 12.2.1 Categories and Genres

151

When Wikipedia contributors create a new article or edit an existing one, among 152 the other things, they assign one or more categories to this article. Categories 153 are used to organize the entire project, and help to give a structure to the 154 whole Wikipedia project by grouping together pages on the same subject. 155 The subcategorization feature makes it possible to organize categories into tree- 156 like structures to help navigation. For example, the movie *Love Actually* belongs 157 to the category *2000s romantic comedy films*, which in turn is a subcategory of 158 *Romantic comedy films* (see Fig. 12.1d). In DBpedia, the hierarchical structure of 159 the categories is maintained through two distinct properties, `dc:terms :subject` 160 and `skos :broader`. The former relates a resource (e.g., a movie) to its categories, 161

while the latter is used to relate a category to its parent categories. Hence, the 162 similarity between two movies can be also discovered in case they have some 163 ancestor categories in common (within the hierarchy). This allows to catch implicit 164 relations and hidden information, that is, information that is not directly detectable 165 looking only at the nearest neighbors in the RDF graph [6]. As an example, thanks 166 to the categories, it is possible to infer a relation between *Love Actually* and *Four 167 Weddings and a Funeral*,<sup>7</sup> since they both belong (indirectly) to the *Romantic 168 comedy films* category. 169

We note that in DBpedia, there is no explicit property about the genre of a 170 movie. At first glance, it might seem quite odd. However, considering what is 171 previously said about categories, in a way they preserve the information about 172 the genre. Furthermore, they are more detailed and subtle than the genres. As an 173 example, if we look at the IMDB page about the movie *Love Actually*,<sup>8</sup> we find that 174 its genres are *Comedy* and *Romance*. Looking at the DBpedia categories for this 175 movie, we find, among the others, *2000s romantic comedy films*, that as previously 176 seen is a subcategory of *Romantic comedy films*. It is easy to see that DBpedia 177 categorization allows a more specific classification of the movie. In order to have 178 both a fine-grained and a standard movie classification, in our approach we also 179 consider the IMDB genres. In fact, DBpedia is not the unique dataset we use 180 for movie similarity detection. We exploit also LinkedMDB [13], that is the RDF 181 version of IMDB, the popular Internet Movie Database, in order to have more robust 182 results. For example, the LinkedMDB URI for the resource referring to the movie 183 *Love Actually* is <http://data.linkedmdb.org/resource/film/39279>. According to the 184 Linked Data principles, resources belonging to this dataset are linked to external 185 datasets, such as DBpedia, by owl:sameAs property. 186

Figure 12.2 shows a sample of the RDF graph containing properties and resources 187 coming both from DBpedia and from LinkedMDB/IMDB. For simplicity, in the 188 picture, we omitted the rdf:type of each node. 189

## 12.3 MORE: More than Movie Recommendation

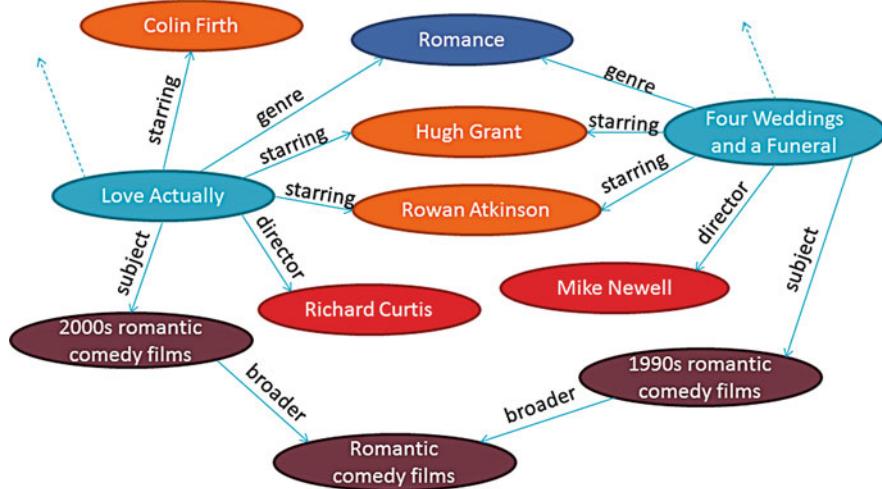
190

In this section, we describe MORE (*MORE than Movie Recommendation*), 191 a Facebook application available online at: [http://apps.facebook.com/movie-193 recommendation/](http://apps.facebook.com/movie-192 recommendation/). A screenshot of the application is depicted in Fig. 12.3. 193

Although the application exploits semantic datasets, the complex semantic nature 194 of the underlying information is hidden to the end user. She does not interact directly 195 with semantic Web languages and technologies such as RDF and SPARQL. The main 196 goals we had in mind during the development of MORE were to build an application 197 (1) intuitive and usable for end users, (2) rich in information taken from different 198

<sup>7</sup>[http://en.wikipedia.org/wiki/Four\\_Weddings\\_and\\_a\\_Funeral](http://en.wikipedia.org/wiki/Four_Weddings_and_a_Funeral)

<sup>8</sup><http://www.imdb.com/title/tt0314331/>



**Fig. 12.2** Sample of RDF graph related to the movie domain

sources (i.e., we wanted to create an original mashup), (3) fast in providing results, 199  
and (4) fully integrated with Web 2.0 social applications as Facebook. 200

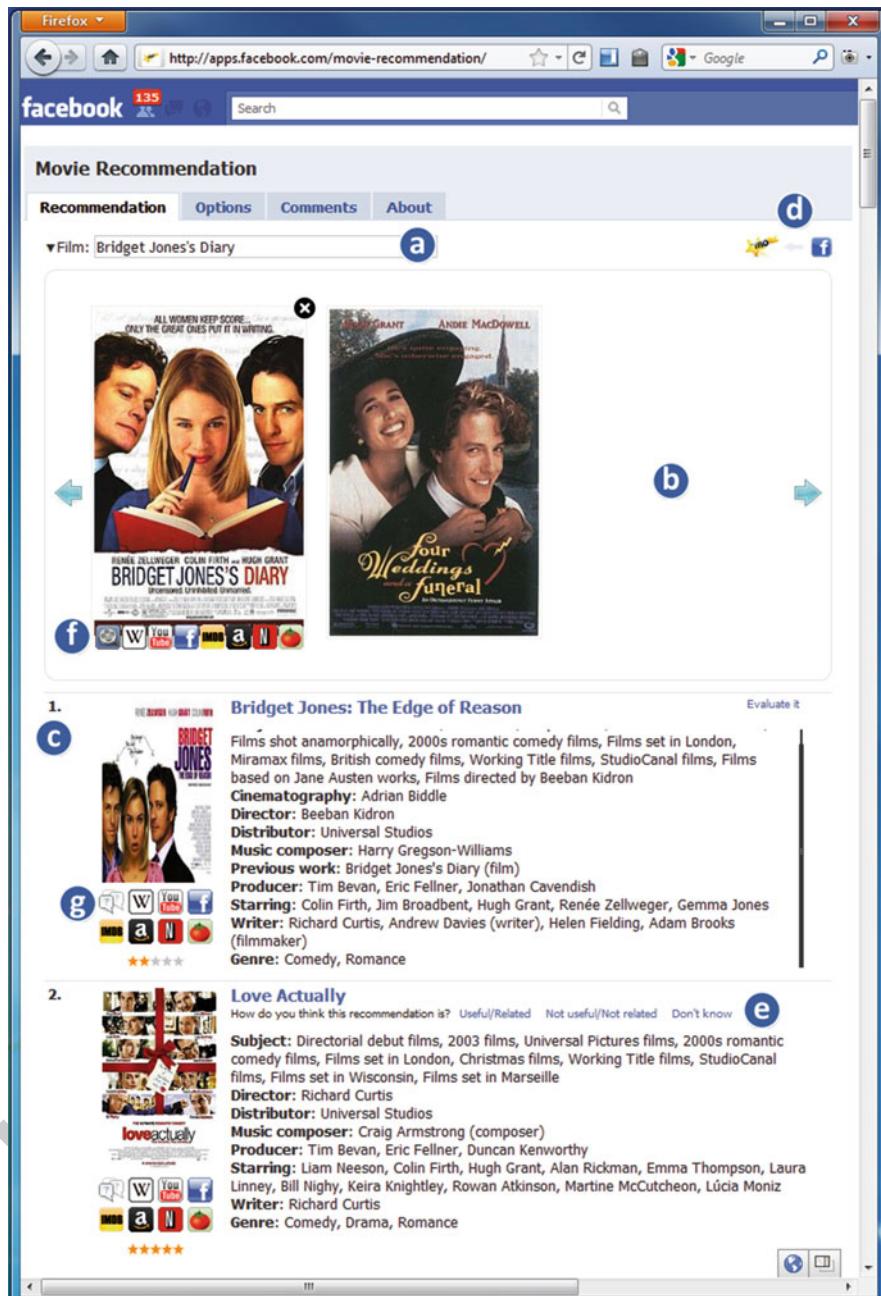
We chose the movie domain mainly for two reasons: (1) movies easily attract 201  
many people that can be considered “domain-expert,” so it is easier to invite people 202  
to use a system and evaluate it, and (2) the resources belonging to the chosen 203  
domain are classified in a consistent DBpedia ontology. Despite the choice of the 204  
movie domain, we stress that, since our system relies on social semantic knowledge 205  
bases, it is potentially able to generate recommendations for any areas covered by 206  
DBpedia. 207

We decided to develop MORE as a Facebook application mainly for the 208  
following three reasons: 209

- Try to minimize the *cold start problem* [23] while recommending new items to 210  
the user. When the user accesses MORE for the first time, the list of movies she 211  
likes is extracted from her Facebook profile. Hence, since the first time MORE 212  
is used, it may already suggest relevant movies to the user by comparing the ones 213  
in her profile with those found in DBpedia. 214
- Improve the quality of recommendations by adopting also a collaborative- 215  
filtering approach. A social network, by its nature, is the best place to implement 216  
a recommender system based on collaborative filtering.<sup>9</sup> 217
- Promote quite easily a new initiative and allow a big part of Internet users to 218  
benefit of the semantic Web innovations [17]. 219

---

<sup>9</sup>This feature is currently not implemented in the online version of the application, and is part of our future work.



**Fig. 12.3** A screenshot of MORE, available online as a Facebook app at <http://apps.facebook.com/movie-recommendation/>

MORE has been designed to be multilingual, exploiting the multilingual nature of 220  
the DBpedia dataset. The language is automatically determined according to the 221  
user language set on Facebook. 222

223

After the application is loaded, the user may search for a *movie* by typing some 224  
characters in the corresponding text field, as indicated by (a) in Fig. 12.3. The 225  
system returns an *autocomplete list* of suggested movies, ranked by popularity. 226  
In order to rank the movies in the *auto-complete list*, we adapted the PageRank 227  
formula [18] to the DBpedia subgraph related to movies. To this aim, we consider 228  
only the property dbpedia-owl:wikiPageWikiLink, which corresponds to 229  
links between Wikipedia pages. In other words, this is equivalent to applying 230  
the PageRank algorithm limited to Wikipedia domain. The idea is that if a 231  
Wikipedia article (i.e., a DBpedia resource) has many incoming links (i.e., many 232  
incoming dbpedia-owl:wikiPageWikiLink relations) from other important 233  
Wikipedia articles, then it is correct to suppose that this article/resource is also 234  
important. In ranking the results shown to the user, we consider also nontopological 235  
information by weighting the results coming from the previous computation with 236  
votes on movies from IMDB users. 237

Once the list has been populated, the user can select one of the suggested movies. 238  
Then, the chosen movie is placed in the user's favorite movies area (see (b) in 239  
Fig. 12.3) and (a) *recommendation* of the top-40 movies related to the selected 240  
one is presented to the user (see c in Fig. 12.3). The relevance rankings for the 241  
movies are computed (off-line) as detailed in Sect. 12.4. The user can add more 242  
movies to her favorite list, just clicking either on its poster or on its title appearing 243  
in the recommendation list. Then, the movie is moved in the favorite area and the 244  
recommendation list is updated, taking into account also the items just added. 245

Another way to add a movie to the favorite list is to exploit the functionalities 246  
offered by the Facebook platform and the Graph API.<sup>10</sup> Facebook users can add 247  
favorite movies to their own Facebook profile. In MORE, the user can obtain her 248  
preferred Facebook movies by clicking on the icon indicated with (d) in Fig. 12.3. 249  
Then, the user can select a movie from the returned list, in order to add it to the 250  
favorite area and to obtain the related recommendation. 251

Each of these actions is tracked by the system. In fact, our goal is to collect 252  
relevant information about user preferences in order to provide a personalized 253  
recommendation that exploits both the knowledge bases such as DBpedia 254  
or LinkedMDB (*content-based* approach) and the similarities among users 255  
(*collaborative-filtering* approach). 256

The properties involved in the similarity detection process do not have the same 257  
importance. For example, the music composer of a movie is *usually* less important 258  
than the actors who played in that movie. Nevertheless, there may be cases where a 259  
user is a fan of a particular music composer and for this reason is interested to all 260  
the movies whose soundtrack is composed by that music composer. Therefore, each 261

---

<sup>10</sup><http://developers.facebook.com/docs/reference/api/>

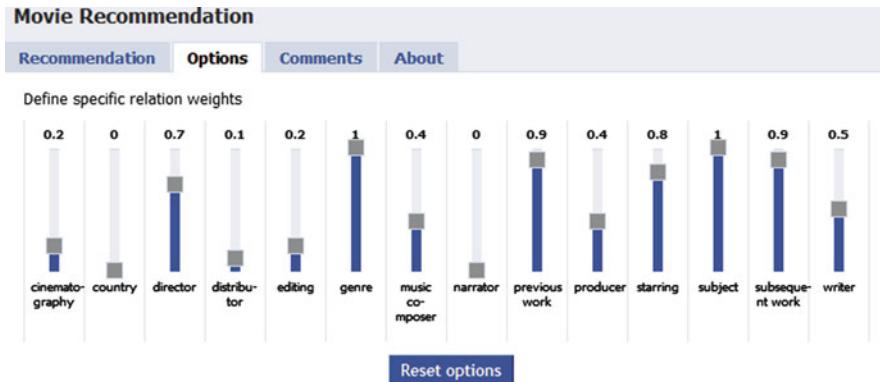


Fig. 12.4 Personalization of the user preferences

property can have a different importance for the user that can be specified through a weight (see Fig. 12.4). In Sect. 12.4, we will detail how we automatically compute the weights associated with each property. However, the user is allowed to set her personal preferences about such properties using the sliders in the *Options* tab, as shown in Fig. 12.4.

If the user moves the mouse over the poster in the favorite movies area, a list of icons fades in (see (f) in Fig. 12.3). Clicking on the first icon on the left (i.e., the one representing a reel), it is possible to obtain *information* about the selected movie, e.g., *starring*, *director*, *subject*. Each of these facets corresponds to a DBpedia property associated with the movie. Each facet can be used to further explore the knowledge associated with the selected movie. The other icons allow to retrieve information related to the selected movie from Wikipedia, YouTube, Facebook, Amazon, IMDB, RottenTomatoes, and Netflix.

For each movie that appears in the recommendation list, in addition to the just mentioned icons, an icon for *explanation* is also available (see (g) in Fig. 12.3). Clicking on it, the user may discover which features the movie shares with the other movies in the favorite area, as shown in Fig. 12.5. In particular, the common values are displayed for each facet (i.e., *subject*, *starring*, etc.).

## 12.4 Semantic Vector Space Model

In order to compute the similarities between movies, we propose a semantic-adaptation of one of the most popular models in classical information retrieval [1]: the vector space model (VSM) [24]. In VSM, nonbinary weights are assigned to index terms in queries and in documents (represented as sets of terms), and are used to compute the degree of similarity between each document in the collection and the query. In our approach, we semanticized the classical VSM, usually used for text retrieval, to deal with RDF graphs.

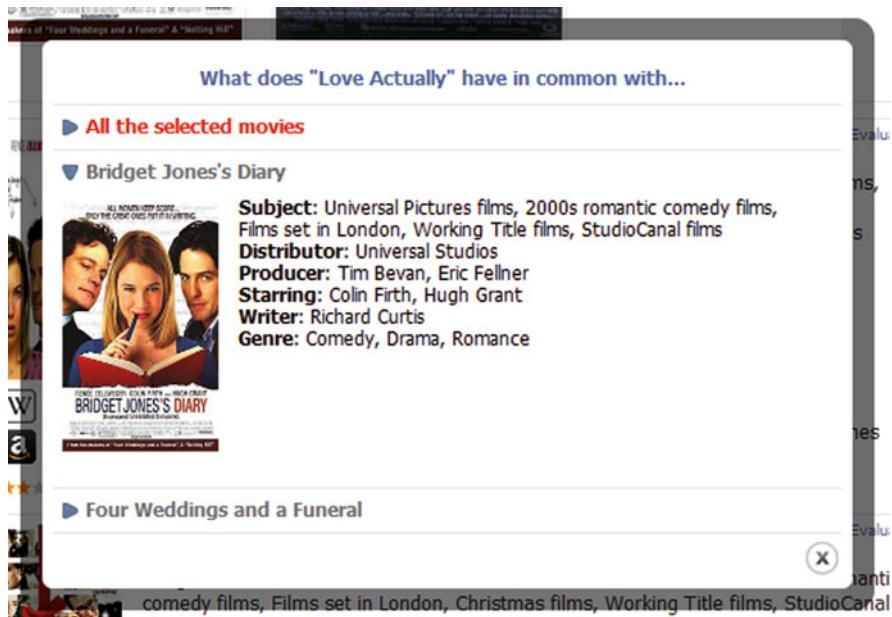


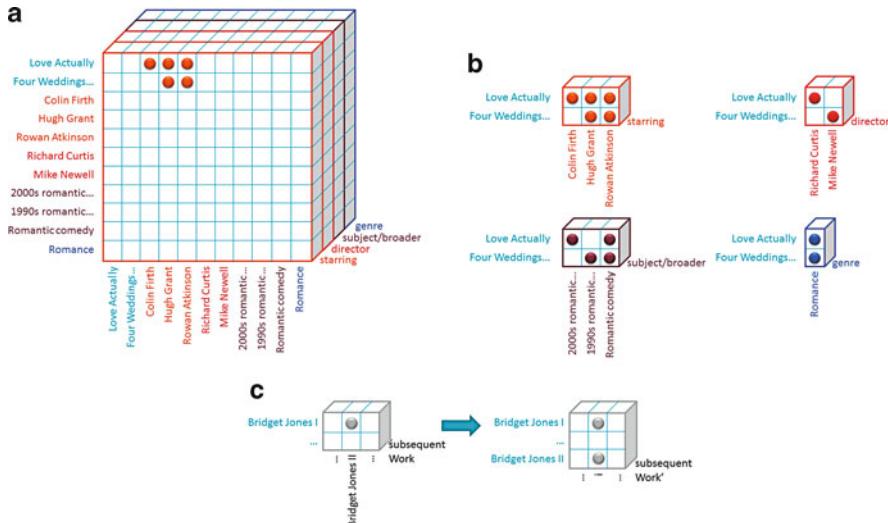
Fig. 12.5 The explanation for the recommended movie

In a nutshell, we represent the whole RDF graph as a three-dimensional tensor where each slice refers to an ontology property. Given a property, each movie is seen as a vector, whose components refer to the *term frequency-inverse document frequency* TF-IDF (or better, in this case, *resource frequency-inverse movie frequency*). For a given slice (i.e., a particular property), the similarity degree between two movies is represented by the correlation between the two vectors, and it is quantified by the cosine of the angle between them. Then, in order to obtain the global correlation between two movies, a weighted sum is calculated considering the similarity values previously computed for each property.

An RDF graph can be viewed as a labeled graph  $G = (V, E)$ , where  $V$  is the set of RDF nodes and  $E$  is the set of predicates (or properties) between nodes in  $V$ . In other words, given the set of triples in an RDF dataset,  $V$  represents the set of all the *subjects* and *objects*. Figure 12.2 shows a sketch of our RDF graph on movies. It contains 2 movies, 3 actors, 2 directors, 3 categories, 1 genre, and 5 different predicates.

In our model, an RDF graph is then a three-dimensional tensor  $T$  where each slice identifies an adjacency matrix for an RDF property (see Fig. 12.6a). All the nodes in  $V$  are represented both on the rows and on the columns. A component (i.e., a cell in the tensor) is not *null* if there is a property that relates a subject (on the rows) to an object (on the columns).

Looking at the model, we may observe and remember that:



**Fig. 12.6** (a) Tensor representation of the RDF graph of Fig. 12.2. Only the components on the first slice (i.e., *starring*) are visible. (b) Slices decomposition. (c) Property transformation

1. The tensor is very sparse. 309
2. Properties are independent of each other (there is no `rdfs:subPropertyOf` relation). 310
3. We are interested in discovering the similarities between movies (or in general between resources of the same `rdf:type` and not between each pair of resources). 312
- 313
- 314

Based on the above observations, we can decompose the tensor slices into smaller matrices. Each matrix of Fig. 12.6b refers to a specific RDF property, and corresponds to a slice in the tensor. In the matrices represented in Fig. 12.6b, movies of the collection are arranged on the rows and represent the *subjects* of RDF triples, while the *objects* of these triples are placed on the columns. In other words, for each matrix, the rows represent the *intensional domain* of the considered property, while the columns its *range*. For a given property, the components of each row represent the contribution of a resource (i.e., an actor, a director, etc.) to the corresponding movie. With respect to a selected property  $p$ , a movie  $m$  is then represented by a vector containing all the terms/nodes related to  $m$  via  $p$ . As for classical information retrieval, the index terms  $k_{n,p}$ , that is, all the nodes  $n$  linked to a movie by a specific property  $p$ , are assumed to be all mutually independent and are represented as unit vectors of a  $t$ -dimensional space, where  $t$  is the total number of index terms. Referring to Fig. 12.6b, the index terms for the *starring* property are *Colin Firth*, *Hugh Grant*, and *Rowan Atkinson*, while  $t = 3$  is the number of all the actors that are objects of a triple involving *starring*.

The representation of a movie  $m_i$ , according to the property  $p$ , is a  $t$ -dimensional vector given by:

$$\overrightarrow{m_{i,p}} = (w_{1,i,p}, w_{2,i,p}, \dots, w_{t,i,p})$$

where  $w_{n,i,p}$  is a nonnegative and nonbinary value representing the weight associated with a term-movie pair  $(k_{n,p}, \overrightarrow{m_{i,p}})$ . The weights  $w_{n,i,p}$  we adopt in our model are TF-IDF weights. More precisely, they are computed as:

$$w_{n,i,p} = f_{n,i,p} * \log\left(\frac{M}{a_{n,p}}\right)$$

where  $f_{n,i,p}$  represents the TF, i.e., the frequency of the node  $n$ , as the object of an RDF triple having  $p$  as property and the node  $i$  as subject (the movie). Actually, this term can be at most 1, since two identical triples cannot coexist in an RDF graph. Then, in case there is a triple that links a node  $i$  to a node  $n$  via the property  $p$ , the frequency  $f_{n,i,p}$  is 1; otherwise,  $f_{n,i,p} = 0$ , and the corresponding weight  $w_{n,i,p}$  is set to 0.

$M$  is the total number of movies in the collection, and  $a_{n,p}$  is the number of movies that are linked to the resource  $n$ , by means of the predicate  $p$ . As an example, referring to Fig. 12.6b, for the *starring* property, and considering  $n = Hugh Grant$ , then  $a_{HughGrant, starring}$  is equal to 2, and it represents the number of movies where *Hugh Grant* acted. The logarithm of the fraction in the formula represents the inverse movie frequency IDF, and as for canonical VSM, it is a global parameter that does not depend on the particular movie  $m_i$ .

Considering all the properties involved in the representation, a movie vector  $\overrightarrow{m_i}$  is defined as:

$$\overrightarrow{m_i} = \frac{1}{P} \sum_p \alpha_p * \overrightarrow{m_{i,p}}$$

where  $P$  is the number of all properties we consider in our model and  $0 \leq \alpha_p \leq 1$  is the importance assigned to each property  $p$ .  $\alpha_p$  can be set by the user or elicited by the system as we will explain at the end of this section. Therefore, each movie can be represented as a  $t \times P$  matrix (it corresponds to a horizontal slice in Fig. 12.6a). If we consider a projection on a property  $p$ , each pair of movies,  $m_i$  and  $m_j$ , is represented as  $t$ -dimensional vector. As for classical VSM, here we evaluate the degree of similarity of  $m_i$  with respect to  $m_j$ , as the correlation between the vectors  $\overrightarrow{m_i}$  and  $\overrightarrow{m_j}$ . More precisely, we calculate the cosine of the angle between the two vectors as:

$$\begin{aligned} \text{sim}(m_i, m_j, p) &= \frac{\overrightarrow{m_{i,p}} \bullet \overrightarrow{m_{j,p}}}{|\overrightarrow{m_{i,p}}| \times |\overrightarrow{m_{j,p}}|} \\ &= \frac{\sum_{n=1}^t w_{n,i,p} * w_{n,j,p}}{\sqrt{\sum_{n=1}^t w_{n,i,p}^2} * \sqrt{\sum_{n=1}^t w_{n,j,p}^2}} \end{aligned}$$

and finally, in order to find the global similarity between  $m_i$  and  $m_j$ , we sum each partial similarity on each property  $p$ : 360  
361

$$\text{sim}(m_i, m_j) = \frac{1}{P} \sum_p \alpha_p * \text{sim}(m_i, m_j, p)$$

where, again,  $\alpha_p$  is the weight associated with each property  $p$ . Thanks to the normalization factor  $P$ , the similarity varies from 0 to 1, where higher values mean higher degree of similarity. 362  
363  
364

Summing up, in the similarity computation process, we firstly compute the similarity for each pair of movies for any property  $p$ , and then we sum all the similarities so obtained for each pair, weighted according to  $\alpha_p$  coefficients. In this way, it is possible to ask the system questions like “Which are the most similar movies to movie  $m_i$  according to the specific property  $p^*$ ?", and also “Which are the most similar movies to movie  $m_i$  according to the whole knowledge base?", and in case “Which are the most similar movies to movie  $m_i$  according to the user profile?" The first question is answered by setting  $\alpha_p = 0, p \neq p^*$ . The second question takes into account the coefficients as computed by the system. The last question considers the user's preferred movies. 365  
366  
367  
368  
369  
370  
371  
372  
373  
374

If we go back to Fig. 12.1, we see that the method described so far to compute the similarity between two movies can be applied to case (b) and (d), i.e., when the similarity has to be found between resources that appear as subjects of RDF triples. Concerning case (c), it is simply a matter of swapping the rows with the columns in the matrices of Fig. 12.6b and applying again the same algorithm. Anyway, we point out that this case is not frequent in movie domain. 375  
376  
377  
378  
379  
380

Concerning case (a) of Fig. 12.1, this allows to discover a similarity between resources that are directly related by some specific properties. In the considered movie domain, this situation happens, for example, with the *subsequentWork* property. In our approach, we operate a matrix transformation to revert this situation to case b) of Fig. 12.1. The transformation is illustrated in Fig. 12.6c. In order to use the VSM with two resources directly linked, the property  $p$  is transformed into the property  $p'$  and its domain remains unchanged. The object of the original RDF triple for the new property  $p'$  is mapped into a unique index associated with the original object (in Fig. 12.6c, index  $i$  is associated with *Bridget Jones II*), and a new RDF triple is created having as subject the original object and as object the index just created. Referring to Fig. 12.6c, *Bridget Jones II* becomes the subject of a new triple, where the predicate is *subsequentWork*' and the object is the index  $i$ . Now our semantic VSM can be applied straight. 381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393

In our implementation, the coefficients  $\alpha_p$  can be set by the user. Anyway, we automatically computed their default values based on a statistical analysis on Amazon's collaborative recommender system. We collected a set of 1,000 randomly selected movies and we checked why users that has bought a movie  $m_i$  also bought movie  $m_j$ , analyzing the first items in the recommendation list. For example, for the movie *Bridget Jones's Diary*, the first movie suggested by Amazon is *Bridget Jones*: 394  
395  
396  
397  
398  
399

*The Edge of Reason.* Then, looking into our semantic dataset, we checked which properties these two movies have in common. In particular, since these movies share part of the cast and some categories and one is the sequel of the other, we assign an initial score to  $\alpha_{\text{starring}}$ ,  $\alpha_{\text{subject/broader}}$ , and  $\alpha_{\text{subsequentWork}}$ . The main idea behind this assignment is that a user likes a movie  $m_i$ , given another movie  $m_j$ , because the two movies are similar according to some properties  $p_1, \dots, p_P$ . More precisely, each time there is a property  $p$  that relates two movies to each other (through any of the configurations in Fig. 12.1), a counter associated with  $p$  is incremented. We iterate the process on the training movie set, and we normalize the counters to the highest one, to finally obtain the coefficients  $\alpha_p$  in the range  $[0, 1]$ .

## 12.5 Evaluation

410

In order to assess the performance and the quality of our recommendation algorithm, we have analyzed the amount of time required to compute the rankings for the whole dataset and we have conducted several experiments with real users to evaluate the proposed recommendation.

411  
412  
413  
414

### 12.5.1 Dataset and Performance Analysis

415

In order to identify the subset of DBpedia resources related to movie domain, we executed some SPARQL queries useful to evaluate its size. More in detail, in the current DBpedia release (3.6), there are 53,619 movies. They appear as the subject of 2,919,686 RDF triples (with 867,966 distinct objects), and as the object of 127,440 RDF triples (with 120,703 distinct subjects). The distinct properties whose domain is a movie are 1,338, while the distinct properties whose object is a movie are 125.

416  
417  
418  
419  
420  
421  
422

Among the almost 3 million of triples having a movie as subject, little more than half of them (1,592,611) link a movie to a resource (e.g., an *actor* URI, a *director* URI, and not a literal). More precisely, there are 188 distinct properties of such type whose domain is a movie.

423  
424  
425  
426

Concerning DBpedia categories, there are 316,483 RDF triples that link a movie to one of the 11,565 distinct categories (via the *dcterms:subject* property). Exploring the dataset one step more, there are 728,796 triples that link each of these categories to a broader category (via the *skos:broader* property).

427  
428  
429  
430

In our analysis, we considered only *object properties* (i.e., properties whose type is *owl:ObjectProperty*), *dcterms:subject* and *skos:broader*, while we discarded both *datatype properties* (i.e., properties whose type is *owl:DatatypeProperty*) and properties that are not mapped in the DBpedia

431  
432  
433  
434

ontology.<sup>11</sup> In particular, we consider `dcterms:subject` and `skos:broader` very relevant to our approach since they encode most of the ontological knowledge in the DBpedia dataset. 435  
436  
437

We agree that considering also literals may improve the recommendation results, 438 but in this work, our focus was just on resources. Nevertheless, we point out that 439 our approach (cf. Sect. 12.4) works also with literals. We use only the properties 440 belonging to the DBpedia ontology, because they allow users to deal with high- 441 quality, clean, and well-structured data. A list of all the properties related to the *Film* 442 class in the DBpedia ontology is available at [443](http://mappings.dbpedia.org/server/)  
[ontology/classes/Film.](#) 444

Focusing on the object properties, DBpedia contains 536,970 triples that link 445 a movie resource to another resource via one of the 16 distinct object properties 446 as defined in the DBpedia ontology. In the movie dataset extraction, we consid- 447 ered all these properties, except for the properties `dbpedia-owl:wikiPage` 448 `ExternalLink` and `dbpedia-owl:thumbnail`, since they clearly do not 449 give any useful semantic contribution to our computation. Just to summarize some of 450 the most relevant characteristics of the extracted movie subgraph, there are 166,844 451 triples involving 52,949 distinct *actors*; 49,538 triples referring to 17,535 different 452 *directors*; and 64,090 triples concerning 28,935 distinct *writers*. 453

Concerning the object properties that link a resource to a movie, the most popular 454 are `dbpedia-owl:subsequentWork` and `dbpedia-owl:previousWork`. 455

After the movie subgraph has been extracted, we have measured the runtime 456 performance of the ranking process executed by our semantic vector space model 457 algorithm. The data collection and filtering steps have been excluded as they depend 458 on the network bandwidth and the availability of the service providing the data. The 459 program is written in Java and makes extensive use of multithreading, with 150 460 concurrent threads. The computation time for the recommendation of the whole 461 extracted dataset has lasted 29 min and 3 s on a dedicated server machine with four 462 Xeon quad-core 2.93GHz processors and 32GB RAM. 463

### 12.5.2 User Evaluation

464

We evaluated our semantic vector space model in two separate rounds with 85 465 different Facebook users per round. We asked them to express a judgment about 466 the recommendation provided by MORE. In the second round, we made some 467 improvements to the algorithm. More precisely, we filtered out some noninformative 468 Wikipedia categories and the categories whose information is already implied by 469 other properties. *English language films* is an example of noninformative category 470 for our application: obviously, two movies do not have to be considered similar 471 just because of the language. Another example of such type of category is *Films* 472

<sup>11</sup><http://wiki.dbpedia.org/Downloads36#ontologyinfoboxproperties>

*whose director won the Best Director Golden Globe:* if the directors of two different movies won a prize, this does not imply that the two movies are similar. We filtered out such noninformative categories using regular expressions and identifying string patterns. We wiped out also another class of categories, such as, for example, the category *Films directed by Jon Avnet*: in this case, the information about the director is already contained within the specific *director* property. If we did not remove it, the information on the director would have been considered twice in the similarity computation. Another example of this type of category is the category *20th Century Fox films*: the distributor is already present as a specific property and has a very low importance. If this category was not eliminated, the discovery of the similarity between movies would have been affected. In order to remove this class of categories, at first we collected the labels associated with directors and distributors, and then we discarded the categories whose label contained the unwanted ones. As shown in Table 12.1, the improved recommendation achieves a precision that is on the average 7.7 percentage points above the baseline (unfiltered) recommendation.

Concerning the evaluation, after users added one or more movies in their favorite movie set, they have been asked to evaluate the proposed movie list. For each of these, the user could vote it as “*useful/related*,” “*not useful/not related*,” or “*don't know*” (cf. (e) in Fig. 12.3). We collected 11,775 votes (on 4,849 distinct movies) in the first round and 11,486 votes (on 4,781 distinct movies) in the second round, meaning that on the average, each user voted 138 times in the first round and 135 times in the second one. The average time spent by each user to express a vote was 9.2 s. The user's favorite movie list contained 1.33 films on the average. From these votes, we calculated the precision in terms of the  $P@n$  [1], with  $n = 1, 3, 5, 10, 40$ , where 40 is the maximum number of recommended movies. This measure provides an assessment of what is the user judgment of the results. The higher the concentration of relevant results at the top of the recommendation list, the more positive the judgment of the users. As noted by Cleverdon in [7], the majority of searches do not require high recall, but just a few relevant answers at the top. The results of the evaluation are shown in Table 12.1. In the evaluation of the precision, we considered as *nonnegative* both the votes marked as “*useful/related*” and the ones marked as “*don't know*.” In our approach, the “*don't know*” vote is very useful and important: it contributes to indicate the novelty of the proposed recommendation. In other words, it can be seen as a kind of serendipitous recommendation, i.e., something new and nonobvious that the user

**Table 12.1** Result of the evaluation: Precision@n

n	Useful		Not useful		Don't know		$P@n$		t48.1 t48.2
	Baseline	Improved	Baseline	Improved	Baseline	Improved	Baseline	Improved	
1	64.4%	76.3%	18.6%	10.2%	17.0%	13.6%	81.4%	89.8%	t48.3
3	57.5%	68.4%	19.7%	12.73%	22.8%	18.8%	80.3%	87.3%	t48.4
5	49.8%	60.6%	24.0%	17.8%	26.3%	21.6%	76.0%	82.2%	t48.5
10	38.0%	49.8%	31.3%	24.7%	30.7%	25.6%	68.7%	75.3%	t48.6
40	34.3%	48.0%	36.5%	26.2%	29.2%	25.7%	63.5%	73.7%	t48.7

would likely not have discovered on her own [25] but she could be interested in. As shown in Table 12.1, the precision decreases when the number of considered results increases. For the first three rows of the table ( $P@1$ ,  $P@3$  and  $P@5$ ), the precision (improved version) is always greater than 80% (considering nonnegative results). It is quite high even for the next rows of the table ( $P@10$  and  $P40$ ), as it is always greater than 70%. 508  
509  
510  
511  
512  
513

Then, we calculated the agreement among testers using the *index raw agreement* [29]. It represents the proportion between the measured interrater agreement and the best possible agreement. The value can vary in the range  $[0, 1]$ , where 1 corresponds to the maximal agreement, while 0 corresponds to no agreement. The results from the evaluation point out a high interrater agreement. In fact, considering the specific agreement, the agreement ratio is 0.87. For the overall agreement, the interrater agreement ratio is even higher, equal to 0.91. The calculated ratios are statistically significant at the 0.05 level (1-tailed). 514  
515  
516  
517  
518  
519  
520  
521

## 12.6 Related Work

522

Several systems have been proposed in literature that address the problem of movie recommendations [12, 14, 16, 21, 25], even if there are very few approaches that exploit the Linked Data initiative to provide semantic recommendation. In the following, we give a brief overview of semantic-based approaches to (movie) recommendation. 523  
524  
525  
526  
527

Szomszor et al. [27] investigate the use of folksonomies to generate tag clouds that can be used to build better user profiles to enhance the movie recommendation. They use an ontology to integrate both IMDB and Netflix data. However, they compute similarities among movies taking into account just similarities between movie tags and keywords in the tag cloud, without considering other information like actors, directors, and writers as we do in MORE. 528  
529  
530  
531  
532  
533

*Filmtrust* [12] integrates semantic Web-based social networking into a movie recommender system. Trust has been encoded using the FOAF trust module and is exploited to provide predictive movie recommendation. *FilmTrust* uses a collaborative filtering approaches as many other recommender systems, as *MovieLens* [14], *Recommendz* [11], and *Film-Consei* [21]. 534  
535  
536  
537  
538

Our RDF graph representation as a three-dimensional tensor has been inspired by [10]. Here, the authors extend the paradigm of two-dimensional graph representation to obtain information on resources and predicates of the analyzed graph. In the preprocessing phase, they just prune dominant predicates, while we automatically extract, through SPARQL queries, relevant ones according to the chosen domain. Moreover, in [10], the authors consider only the objects of the RDF triples, while we look also at subject of the statements to compute similarities. 539  
540  
541  
542  
543  
544  
545

Tous and Delgado [28] use the vector space model to compute similarities between entities for ontology alignment; however, with their approach, it is possible to handle only a subset of the cases we consider, specifically only the case where 546  
547  
548

resources are directly linked (see Fig. 12.1a). Eidon et al. [8] represent each concept 549  
in an RDF graph as a vector containing nonzero weights. However, they take into 550  
account only the distance from concepts and the subclass relation to compute such 551  
weights. 552

Effective user interfaces play a crucial role in order to provide a satisfactory 553  
user experience for content recommendation. Nowadays, there are some initiatives 554  
that exploit the Linked Data cloud to provide effective recommendations. One 555  
of these is *dbrec* [19], a music content-based recommender system that adopts 556  
an algorithm for *Linked data semantic distance* [20]. It uses only DBpedia as 557  
knowledge base in the Linked Data cloud. The recommendation is link-based, 558  
since the “semantics” of relations is not exploited and each relation has the same 559  
importance, and it does not take into account the links hierarchy, expressed in 560  
DBpedia through the SKOS vocabulary. 561

## 12.7 Conclusion and Future Work

562

The huge amount of Linked Data freely available in the so-called Web of Data are 563  
for sure an interesting opportunity to build new and smarter knowledge-based appli- 564  
cations. In this chapter, we have presented MORE, a Facebook application that 565  
works as a recommender system in the movie domain. The background knowledge 566  
adopted by MORE comes exclusively from semantic datasets. In particular, in this 567  
version of the tool, we use DBpedia and LinkedMDB to collect information about 568  
movies, actors, directors, etc. The recommender algorithm relies on a semantic ver- 569  
sion of the classical vector space model adopted in information retrieval. Currently, 570  
we are working on the next version of the tool to include both technological and 571  
methodological improvements. From a technological point of view, we are testing 572  
the performances of the recommendation algorithm when using NoSQL databases 573  
[26]. In particular, due to the graph-based nature of the information we manage, 574  
we are working on an implementation based on the graph database Neo4j.<sup>12</sup> 575  
Moreover, we are willing to better integrate MORE in the Linked Data cloud by 576  
publishing our recommendation using the *recommendation ontology* [9]. Following 577  
the idea of Tim Berners-Lee et al. in [2]: “*one of the goals of a semantic browser* 578  
*is serendipitous re-use*,” we want to investigate how exploratory browsing [30] 579  
might be helpful when combined with a recommender system for serendipitous 580  
recommendations [15]. We are also developing a mobile version of the system which 581  
exploits location-based information both for the exploratory search task and for 582  
the recommendation process. From a methodological perspective, we are collecting 583  
information from MORE users to implement also a collaborative-filtering approach to 584  
recommendation. This is particularly relevant and challenging since the application 585  
is integrated with Facebook. 586

---

<sup>12</sup><http://neo4j.org>

## References

1. Baeza-Yates, R.A., Ribeiro-Neto, B.: Modern Information Retrieval: The Concepts and Technology Behind Search. Addison-Wesley Professional, Reading, MA (2011) 588
2. Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Lerer, A., Sheets, D.: Tabulator: exploring and analyzing linked data on the semantic web. In Proceedings of the 3rd international semantic web user interaction workshop (SWUI06) (2006) 590
3. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web: scientific American. Scientific American (2001) 593
4. Bizer, C., Heath, T., Berners-Lee, T.: Linked data – the story so far. Int. J. Semant. Web Inf. Syst. **5**(3), 1–22 (2009) 595
5. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: Dbpedia – a crystallization point for the web of data. Web Semant. **7**, 154–165 (2009) 597
6. Chernov, S., Iofciu, T., Nejdl, W., Zhou, X.: Extracting semantics relationships between wikipedia categories. In: Volkel M., Schaffert S. (eds.) Proceedings of the 1st Workshop on Semantic Wikis – from Wiki to Semantics, Workshop on Semantic Wikis. ESWC2006 (2006) 599
7. Cleverdon, C.W.: The significance of the cranfield tests on index languages. Proceedings of the 14th Annual International ACM SIGIR, pp. 3–12 (1991) 602
8. Eidoon, Z., Yazdani, N., Oroumchian, F.: A vector based method of ontology matching. Proceedings of 3rd International Conference on Semantics, Knowledge and Grid, pp. 378–381 (2007) 604
9. Ferris, B., Jacobson, K.: The Recommendation Ontology 0.3. [http://purl.org/ ontology/rec-core#](http://purl.org/ontology/rec-core#) (2010) 607
10. Franz, T., Schultz, A., Sizov, S., Staab, S.: Triplerank: ranking semantic web data by tensor decomposition. Proceedings of the 8th ISWC, ISWC '09, pp. 213–228 (2009) 609
11. Garden, M., Dudek, G.: Semantic feedback for hybrid recommendations in recommendz. IEEE international conference EEE'05, pp. 754–759 (2005) 611
12. Golbeck, J., Hendler, J.: Filmtrust: movie recommendations using trust in web-based social networks. Proceedings of the IEEE CCNC (2006) 613
13. Hassanzadeh, O., Consens, M.P.: Linked movie data base. Proceedings of the WWW2009 Workshop on Linked Data on the Web (LDOW2009) (2009) 615
14. Herlocker, J., Konstan, J.A., Riedl, J.: Explaining collaborative filtering recommendations. Proceeding on the ACM 2000 Conference on Computer Supported Cooperative Work, pp. 241–250 (2000) 617
15. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. ACM Trans. Inf. Syst. **22**, 5–53 (2004) 620
16. Mukherjee, R., Sajja, N., Sen, S.: A movie recommendation system – an application of voting theory in user modeling. User Model. User-Adapt. Interact. **13**(1–2), 5–33 (2003) 622
17. Nazir, A., Raza, S., Chuah, C.N.: Unveiling facebook: a measurement study of social network based applications. Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement, IMC '08, pp. 43–56. ACM, New York, NY, USA (2008) 624
18. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: bringing order to the web. Technical Report 1999-66, Stanford InfoLab (1999) 627
19. Passant, A.: dbrec: music recommendations using dbpedia. Proceedings of the 9th International Sematic Web Conference, ISWC'10, pp. 209–224 (2010) 629
20. Passant, A.: Measuring semantic distance on linking data and using it for resources recommendations. Proceedings of the AAAI Spring Symposium "Linked Data Meets Artificial Intelligence" (2010) 631
21. Perny, P., Zucker, J.: Preference-based search and machine learning for collaborative filtering: the film-consei recommender system. Inform. Interac. Intel. **1**, 9–48 (2001) 634
22. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Recommendation (2008) 636

23. Ricci, F., Rokach, L., Shapira, B., Kantor, P.B. (eds.): *Recommender Systems Handbook*. Springer, Berlin, Heidelberg, New York (2011) 638  
639
24. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. *Commun. ACM* **18**, 613–620 (1975) 640  
641
25. Schafer, J.B., Konstan, J.A., Riedl, J.: E-commerce recommendation applications. *Data Mining Knowledge Dis.* **5**, 115–153 (2001) 642  
643
26. Stonebraker, M.: Sql databases v. nosql databases. *Commun. ACM* **53**, 10–11 (2010) 644
27. Szomszor, M., Cattuto, C., Alani, H., O'Hara, K., Baldassarri, A., Loreto, V., Servedio, V.D.: Folksonomies, the semantic web, and movie recommendation. 4th European semantic web conference (2007) 645  
646  
647
28. Tous, R., Delgado, J.: A vector space model for semantic similarity calculation and owl ontology alignment. *DEXA*, pp. 307–316 (2006) 648  
649
29. Von Eye, A., Mun, E.Y.: *Analyzing Rater Agreement: Manifest Variable Methods*. Lawrence Erlbaum Associates, Mahwah, NJ, USA (2004) 650  
651
30. White, R.W., Roth, R.A.: Exploratory search: beyond the query-response paradigm. *Syn. Lect. Inform. Concepts Retriev. Serv.* **1**(1), 1–98 (2009) 652  
653

# Chapter 13

## Flint: From Web Pages to Probabilistic Semantic Data

Lorenzo Blanco, Mirko Bronzi, Valter Crescenzi, Paolo Merialdo,  
and Paolo Papotti

### 13.1 Introduction

The Web is a surprisingly extensive source of information: it offers a huge number of sites containing data about a disparate range of topics. Although Web pages are built for human fruition, not for automatic processing of the data, we observe that an increasing number of Web sites deliver pages containing structured information about recognizable concepts, relevant to specific application domains, such as movies, finance, sport, products, etc.

The development of scalable techniques to *discover*, *extract*, and *integrate* data from fairly structured large corpora available on the Web is a challenging issue, because to face the Web scale, these activities should be accomplished automatically by domain-independent techniques.

To cope with the complexity and the heterogeneity of Web data, state-of-the-art approaches focus on information organized according to specific patterns that frequently occur on the Web. Meaningful examples are WebTables [19], which focuses on data published in HTML tables, and information extraction systems, such as TextRunner [4], which exploits lexical-syntactic patterns. As noticed by Cafarella et al. [19], even if a small fraction of the Web is organized according to these patterns, due to the Web scale, the amount of data involved is impressive.

In this chapter, we focus on methods and techniques to wring out value from the data delivered by large data-intensive Web sites. These sources are characterized by

---

AQ1

L. Blanco (✉) · M. Bronzi · V. Crescenzi · P. Merialdo · P. Papotti

Dipartimento di Informatica e Automazione, Università degli Studi Roma Tre, via della Vasca Navale 79, Rome, Italy

e-mail: [blanco@dia.uniroma3.it](mailto:blanco@dia.uniroma3.it); [bronzi@dia.uniroma3.it](mailto:bronzi@dia.uniroma3.it); [crescenz@dia.uniroma3.it](mailto:crescenz@dia.uniroma3.it);  
[merialdo@dia.uniroma3.it](mailto:merialdo@dia.uniroma3.it); [papotti@dia.uniroma3.it](mailto:papotti@dia.uniroma3.it)

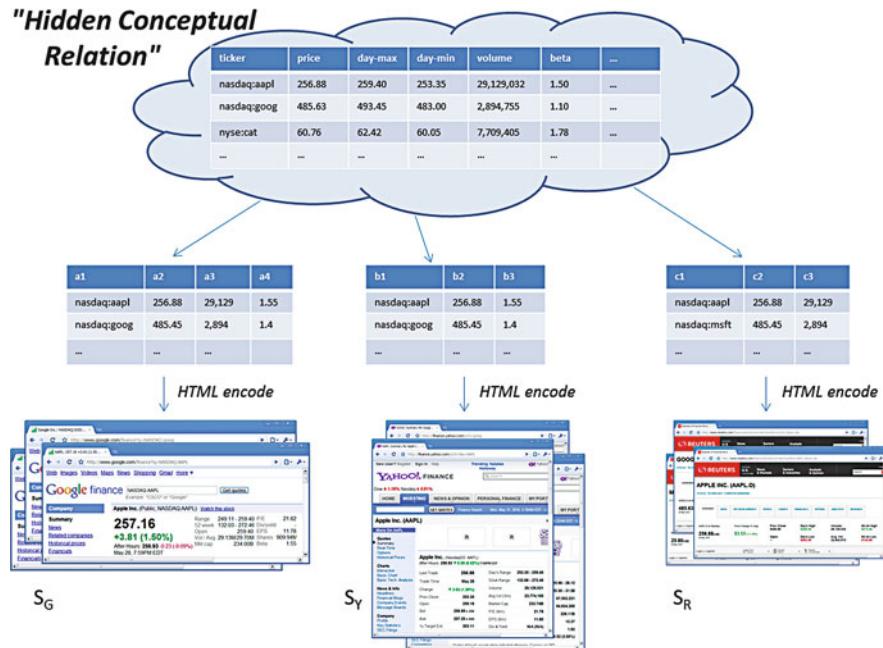
INTL BUSINESS MACH (NYSE: IBM)		Cisco Systems, Inc. (CSCO.O)			Google finance		
		sector: Technology, Industry: Communications Equipment			NASDAQ:AAPL Example: "CSCO" or "Google"		
Last Trade:	<b>118.76</b>	Day's Range:	<b>118.16 - 119.00</b>	Price	<b>22.93</b> USD	Price Change	<b>+0.13</b>
Change:	<b>+0.12 (0.10%)</b>	\$2wk Range:	<b>69.50 - 124.00</b>	Percent Change			<b>+0.57%</b>
Prev Close:	<b>118.68</b>	Volume:	<b>1,415,704</b>	Last Trade	<b>\$22.92</b>	Day's High	<b>\$22.98</b>
Open:	<b>118.78</b>	Avg Vol (3m):	<b>6,578,780</b>	Change	<b>+0.57%</b>	Day's Low	<b>\$22.68</b>
Bid:	<b>N/A</b>	Market Cap:	<b>155.6B</b>	Prev Close	<b>\$22.79</b>	52-wk High	<b>\$24.30</b>
Ask:	<b>N/A</b>	P/E (ttm):	<b>12.68</b>	Open	<b>\$22.87</b>	52-wk Low	<b>\$13.61</b>
1y Target Est:	<b>127.15</b>	EPS (ttm):	<b>9.368</b>	Volume	<b>18,750,655</b>	Beta	<b>1.20</b>
		Div & Yield:	<b>2.20 (1.90%)</b>			Avg. Vol	<b>48,902,344</b>

Fig. 13.1 Three Web pages containing data about stock quotes from Yahoo!, Reuters, and Google

two important features that suggest new opportunities for the automatic discovery, 27  
extraction, and integration of Web data. 28

- On the one hand, we observe *local regularities*: in these sites, large amounts 29  
of data are usually offered by hundreds of pages, each encoding one tuple in a 30  
local HTML template. For example, pages shown in Fig. 13.1 (which are from 31  
three different Web sites) publish information about one company stock. If we 32  
abstract this representation, we may say that each Web page displays a tuple, 33  
and that, in a given Web site, the collection of pages containing data about the 34  
same entity corresponds to a relation. According to this abstraction, the web sites 35  
for pages in Fig. 13.1 expose their own “StockQuote” relation. Local regularities 36  
also involve the access paths to reach pages offering data from the same relation. 37  
In our example, within each site, the navigation paths to reach stock quote pages 38  
from, say, the home page are similar. 39
- On the other hand, we notice *global information redundancy*: many sources 40  
provide similar information. For example, there are dozens of Web sites that 41  
publish pages containing stock quote data. Global information redundancy occurs 42  
both at the schema level (the same attributes are published by more than one 43  
source) and at the extensional level (some objects are published by more than 44  
one source). In our example, at the schema level, many attributes are present 45  
in all the sources (e.g., the company name, last trade price, volume); while 46  
others are published by a subset of the sources (e.g., the “Beta” indicator). 47  
At the extensional level, there is a set of stock quotes that are published by more 48  
sources. It is worth observing that, as web information is inherently imprecise, 49  
redundancy also implies inconsistencies; that is, sources can provide conflicting 50  
information for the same object (e.g., a different value for the volume of a given 51  
stock). 52

The above observations lead us to abstract that underlying sources of the same 53  
domain there is a *hidden conceptual relation* from which collections of pages from 54  
different Web sites are generated. According to this model, each collection can be 55  
seen as a source of data created by a generative process applied over the hidden 56  
relation. Each source publishes information about a subset of the tuples from the 57  
hidden relation, and different sources may publish different subsets of its attributes. 58  
Sources may introduce errors and imprecisions, or they may publish values by 59  
adopting different formats (e.g., miles vs. kilometers). Finally, each source encodes 60



**Fig. 13.2** The generative process applied to the publication of stock quote pages by GoogleFinance, YahooFinance, and Reuters

data in a local HTML template, giving rise to the published collection of pages. 61  
 Figure 13.2 depicts this process applied to three finance Web sources that publish 62  
 their own views of the *hidden relation*. 63

### 13.1.1 Inverting the Publishing Process

This chapter describes the main results of a research project, called FLINT, that 65  
 aims at developing methods, tools, and techniques to support the construction of 66  
 repositories from Web data by leveraging local regularities and global information 67  
 redundancy. Given an entity of interest (such as, stock quote, book, soccer player), 68  
 we have studied how to automatically *discover* sites that offer large collections of 69  
 pages publishing data about that entity. Based on the hidden relation model, we 70  
 have developed techniques to automatically *extract and integrate* the data offered 71  
 by these pages. Finally, since Web data are inherently imprecise, and conflicting 72  
 values usually arise from different sources, we have applied a Bayesian framework 73  
 to *assess the quality* of the integrated data and to evaluate the accuracy of the 74  
 sources. 75

### 13.1.2 Chapter Outline

76

Section 13.2 describes the FLINT approach to discover sources publishing data about a given target entity. Then, Sect. 13.3 illustrates how the gathered pages are processed to extract and integrate data. Section 13.4 presents models and methods to compute a probability distribution for the values of the integrated data and the accuracy of the sources. The results of some experiments are described in Sect. 13.5, and related work is discussed in Sect. 13.6. Finally, Sect. 13.7 concludes the chapter.

## 13.2 Discovering New Sources

83

Our approach concentrates on sites that publish information according to the hidden relation model introduced in the previous section. This section describes methods to discover and gather large collection of pages containing data that represent instances of a given conceptual entity of interest.

In certain domains, such as the finance one, the set of instances is well established (the stocks quoted in the largest markets) and the many authoritative Web sources are known. In these cases, a small repository of identifiers for the relevant entity can be already available or easily built (the set of the stock quote tickers can be easily obtained), and then the collection of pages can be effectively retrieved by automatically filling a search Web form within the sites with the already known identifiers (sites that offer financial information usually provide a form that accepts as input the stock ticker and that returns as output a page containing the details about the corresponding stock).

In other domains, the set of instances and their identifiers, as well as the set of sites and the methods to access the collection of relevant pages that they offer, are not known *a priori*. To overcome these issues, we have developed a method to discover large collections of pages that publish data about an entity of interest.

Our method leverages both global information redundancy and local regularities. On the one hand, it relies on the redundancy of information to discover and select new sources [11]; on the other hand, it exploits the local regularities of pages and navigational paths to collect the target pages delivered by the selected sites [10].

Our method takes as input a bunch of sample pages, each containing data about one instance of an entity of interest, and returns collections of pages that publish data about instances of that entity. For example, pages such as those in Fig. 13.1 can be used as input to harvest collections of pages containing company stock quote data. The overall approach can be summarized by the following three main steps:

- *Local crawling*: we crawl the Web sites of the input sample pages to collect pages with data about other instances of the entity of interest.
- *Description extraction*: from the collected pages, we automatically extract a set of keywords that represent an intensional description of the entity exemplified by the sample pages.

- *Search expansion:* using the information computed in the previous steps, we launch Web searches to discover new sources. The results of these searches are analyzed using the entity description. Pages representing valid instances of the target entity are collected and used to recursively trigger the process.

Before giving more details about the three steps, observe that our technique has a different semantics with respect to the “similar pages” facility offered by search engines. Given as input two Web pages from two different Web sites describing the basketball players “Kobe Bryant” and “Bill Bradley,” our method aims at retrieving many Web pages that are similar at the intensional level, e.g., pages about other basketball players, not necessarily the same two sample players.

### 13.2.1 Local Crawling: Searching Entity Pages Within One Site

The first step of our method aims at searching the target pages within the Web sites that publish the input sample pages. To achieve this goal, one could cluster the pages of the whole Web sites (e.g., by means of techniques such as those proposed in [15]) and then select the clusters containing the sample pages. However, since the target pages may represent a small subset of the whole site, in order to avoid the costs of downloading a large number of pages, we have developed a structure-driven crawler, specifically tailored to accomplish this goal.

Our structure-driven crawler relies on the observation that, for the sake of scalability of the publishing process, data-intensive Web sites publish pages where the structure and the navigation paths are fairly regular. Within each site, pages containing the same *intensional* information, i.e., instances of the same entity, organize data according to a common template, and the access paths (e.g., from the home page) to these pages obey to a common pattern. Therefore, given a seed page containing data of interest, the goal of the structure-driven crawler is to navigate the seed page’s site in order to pick out the largest number of pages similar in structure to the seed pages.

To model the structure of a Web page, we rely on few features that nicely abstract the template of a Web page. Our model considers that pages from large Web sites usually contain a large number of links, and pages generated by the same template have links that share the same layout and presentation properties. Therefore, whenever a large majority of the links of two pages have the same layout and presentation properties, it is likely that the two pages have been generated by the same template, that is they are similar in structure. Then, the structure of a Web page is described by means of the presentation and layout properties of the links that it offers, and the structural similarity between pages is measured with respect to these features.

Another important characteristic of data-intensive Web sites is that collections of links that share the same layout properties (consider, e.g., a list of links in one table column) usually lead to pages that contain similar information.

Based on these observations, the crawler explores the Web site to find out which pages contain lists of links toward pages which are structurally similar to the seed pages. Following these access paths, it gathers all the site's pages that are structurally similar to the seed pages.

Beside collecting these pages, the crawler also extracts the anchors of the links leading to them. As we will describe later in the chapter, the role of these anchors is very important in our methods, since they are used as identifiers for the conceptual instances published in the referenced pages. In our context, in which pages contain data about one entity of interest, it is likely that the anchor identifies the target object. In our financial example, the anchor of the links to each stock quote page usually corresponds to the name of the company stock quote. It is worth observing that, for the sake of usability, this feature has a general validity on the Web. For example, the anchor to a book page usually corresponds to the book title, the anchor to a football player page usually is the player name, and so on.

### 13.2.2 Learning a Description of the Entity

169

During the second step, our approach computes a description for the target entity. The description of an entity is expressed as a set of *weighted intensional keywords*. For instance, a possible description for the stock quote entity is the set of keywords: *price, volume, high, and low*, all with equal weights.

Given a set of structurally similar pages from the same site as returned by the structure-driven crawler, our method relies on the observation that pages containing data about instances of the same entity share a common set of characterizing terms among those appearing in the page template.

Therefore, the entity description is computed as the set of terms that more frequently occurs in the pages returned by the structure-driven crawler for a given Web site. However, not all these terms can be used to characterize the entity, because there is significant amount of noise due to the presence of template terms that do not describe the underlying entity. These are terms that usually belong to a site-scope template, not specific of the pages collected by the crawler; e.g., footers and headers are shared by all the pages in the site.

Noisy terms are filtered by considering multiple Web sites at once: terms shared in several templates of different Web sites are selected as keywords of the entity description and weighted according to the frequency of appearance.

### 13.2.3 Search Expansion

188

The results produced in the first two steps are used to propagate the search on the Web. This step is done by issuing a set of queries against a search engine and elaborating the results in order to select only those pages that can be considered

189

190

191

as instances of the target entity. The selected pages are then used as seeds to trigger a new execution of the structure-driven crawler, which gathers new collections of pages. If the discovered collections contain new instances, these are used to launch new searches, and the whole process is repeated until new pages can be found at a reasonable pace.

To feed the search engine with suitable keywords, we use the identifiers associated with the pages returned by the structure-driven crawler. That is, we pose a number of queries against a search engine, where each query is composed by the anchor of a link to one of the pages retrieved by the previous structure-driven crawler execution.

Search results can include also pages that are not suitable for our purposes. Typical examples are pages from forums, blogs, or news where the keywords occur by chance, or because they are in a free-text description. To control this aspect, our method requires that the keywords of the entity description appear in the template of the retrieved page.

Then, for each page returned by the search engine, an instance of the structure-driven crawler is run to obtain a set of structurally similar pages, and their template is computed. If the computed template contains the keywords of the entity description, the page is considered valid; otherwise, it is discarded.

Valid pages are finally used as seeds for new structure-driven crawler scans, thus contributing to further discover new pages.

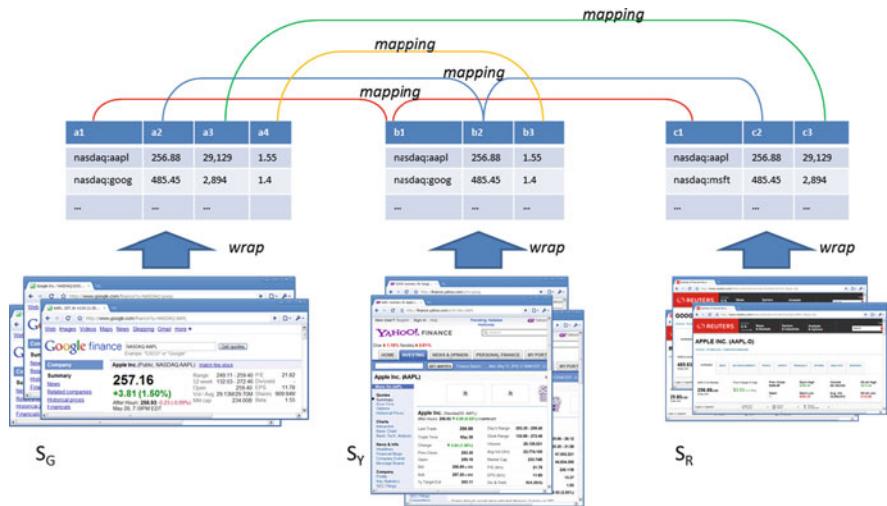
### 13.3 Extracting and Integrating Web Data

213

We now describe our approach to extract and integrate data from the sources. In our context, sources consist of collections of pages, where each page contains data about one instance of a given entity. Also, as we have discussed in Sect. 13.2, our source discovery method associates every page with a string that works as an identifier for the underlying instance.

According to the hidden relation model introduced in Sect. 13.1, the issue of extracting and integrating data from the sources corresponds to invert the page generation process, i.e., to build the hidden relation starting from the collections of pages.

A natural solution to the problem is a two-step “waterfall” approach. Figure 13.3 depicts the process. First, a wrapper generation tool produces wrappers to extract data from the pages; the application of the wrappers over the collections of pages returns a set of relations. Then, a schema matching algorithm integrates the relations by creating mappings among attributes with the same semantics. However, such an approach does not scale when a large number of sources are involved, and a high level of automation is required. In fact, to face the Web scale, wrappers should be generated automatically by an unsupervised process. But automatic wrapper generators can produce imprecise extraction rules (e.g., rules that extract irrelevant or wrong data), and to obtain correct rules, the wrappers should be evaluated and



**Fig. 13.3** Inverting the page generation process: automatically generated wrappers extract relations from the pages; a schema matcher creates mappings that group semantically homogeneous mappings

refined manually. Moreover, the relations extracted by automatically generated 233  
wrappers are opaque, i.e., their attributes are not associated with any (reliable) 234  
semantic label. Therefore, the matching algorithm should rely on instance-based 235  
approaches, which consider only attribute values to match schemas. Unfortunately, 236  
instance-based matching is challenging because the sources may provide conflicting 237  
values (due to the errors introduced by the publishing process), and imprecise 238  
extraction rules may return wrong, and thus inconsistent, data. 239

To overcome these issues, we have developed an automatic, domain-independent 240  
approach that exploits the redundancy of data among the sources to support both 241  
the extraction and the matching steps. In a bootstrapping phase, an unsupervised 242  
wrapper inference algorithm generates a set of extraction rules for each source. 243  
A domain-independent, instance-based matching algorithm compares data returned 244  
by the generated extraction rules among different sources and infers mappings. The 245  
abundance of redundancy among Web sources allows us to acquire knowledge about 246  
the domain and triggers an evaluation of the mappings. Based on the quality of 247  
the inferred mappings, the matching process provides a feedback to the wrapper 248  
generation process, which is thus driven to refine the bootstrapping wrappers in 249  
order to correct imprecise extraction rules. Better extraction rules generate better 250  
mappings, thus improving the overall quality of the solution. At the end of the 251  
process, we also rely on the evidence accumulated in order to extract from the page 252  
templates suitable semantic labels for the mappings, thus associating a semantic 253  
label with the integrated data. 254

In the following, we provide a description of the approach. The interested reader 255  
can find technical details in [8, 13]. 256

### 13.3.1 Contextual Data Extraction and Integration

257

In our context, a wrapper is an ordered set of extraction rules that apply over a 258  
 AQ3 Web page. Each each rule extracts a (possibly empty) string from the HTML of 259  
 the page. Therefore, the application of a wrapper over a page returns a tuple, and 260  
 the application of a wrapper over the collection of pages of a source returns a 261  
 relation, whose schema has as many attributes as the number of extraction rules 262  
 of the corresponding wrapper. 263

In a bootstrapping phase, for each source an automatic wrapper inference system 264  
 (such as RoadRunner [24] or ExAlg [3]) generates a wrapper. As the process is 265  
 unsupervised, the wrappers can contain *weak* rules, i.e., rules that do not correctly 266  
 extract data for all the source pages, and *useless rules*, i.e., rules that extract 267  
 irrelevant data. As we shall discuss later, weak rules are refined, and useless rules 268  
 are identified (and discarded) contextually with the integration process, leveraging 269  
 the redundancy of information. 270

The attributes of the relations produced by the wrappers are processed in order to 271  
 produce *mappings*, i.e., sets of attributes with the same semantics. Since wrappers 272  
 produce anonymous relations, mappings are computed by means of a *distance* 273  
*function* among the values of a small sample set of *aligned* tuple pairs. Two tuples 274  
 are aligned if they refer to the same real-world object. In our framework, the 275  
 task of aligning tuples is simplified by the presence of the identifiers returned by 276  
 the structure-driven crawler: we align tuples that are extracted from pages having 277  
 the same identifiers. The distance function returns a score between 0 and 1: the 278  
 more similar are two attributes, the lower is the distance. 279

A simple method to infer the mappings consists in applying a variant of standard 280  
 hierarchical agglomerative clustering [38] approaches. Starting from a pivot source, 281  
 we build one singleton mapping for every attribute. Then, we iterate over the other 282  
 relations: for each attribute of the current relation, we determine the mapping with 283  
 lowest distance.<sup>1</sup> If such a distance is lower than a given threshold, the attribute is 284  
 added to the mapping. Whenever an attribute is not added to an existing mapping, 285  
 it gives rise to a new singleton mapping: it might represent an attribute that was not 286  
 present in the sources processed so far. 287

At the end of the iteration, singleton mappings are discarded. Assuming that 288  
 relevant attributes are published by at least two sources, singleton mappings 289  
 represent information extracted by useless rules. 290

The described method is limited by the strong dependence on the matching 291  
 threshold, which is statically assigned to every comparison. Low values of the 292  
 threshold tend to generate many small mappings, because small imprecisions are 293  
 not tolerated. Conversely, high values produce large heterogeneous mappings, 294  
 composed of attributes with different semantics. The choice of a threshold that 295  
 represents a nice trade-off between precision and recall is not trivial. The financial 296

---

<sup>1</sup>The distance between an attribute and a mapping is from the centroid of the mapping.

scenario introduced in the previous examples represents an interesting example 297  
 that clearly explains the problem. Observe that several attributes exhibit values 298  
 that are very close to one another: this is the case for the minimum, average, 299  
 and maximum price of a stock quote. For these attributes, low threshold values 300  
 are needed. Conversely, for other attributes, such as the volume or the market 301  
 capitalization, higher threshold values are preferable since for these values, sources 302  
 are less precise. 303

To address these issues, we have developed a clever approach, called SplitAnd- 304  
 Merge, that dynamically computes the matching threshold for every mapping. 305

### 13.3.1.1 SplitAndMerge Matching

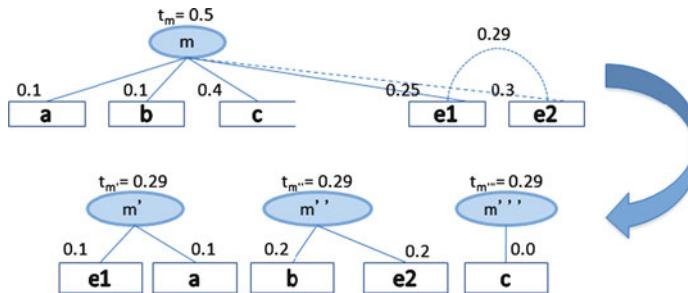
*SplitAndMerge* is based on the observation that it is unlikely that a source publishes 307  
 the same attribute more than once, with different values. We therefore assume that 308  
 nonidentical attributes from the same source have always different semantics. As a 309  
 consequence, we impose the constraint, called *different semantic constraint*, that a 310  
 mapping is not allowed to include more than one attribute from the same source. 311

Also in *SplitAndMerge* mappings are created iterating over the source relations. 312  
 However, each mapping is associated with its own specific threshold. Whenever a 313  
 new mapping is created from scratch, it is associated with a given threshold, but the 314  
 threshold is dynamically updated during the process whenever the different semantic 315  
 constraint is violated. 316

As in the static threshold approach, an attribute can be added to a mapping if 317  
 its distance from the mapping is lower than the mapping threshold. Nevertheless, 318  
 before adding an attribute to a mapping, we check whether the mapping already 319  
 contains another attribute from the same source. Clearly, if two attributes from the 320  
 same source match a mapping, their distances from the mapping are lower than 321  
 the threshold. However, as such a threshold value would lead to a violation of the 322  
 different semantics constraint, we can conclude that the two attributes represent 323  
 different concepts, and that the values of these concepts are more similar than 324  
 expected by the current threshold, which is too high for that mapping. We observe 325  
 that a suitable threshold value should keep well separated the two attributes that led 326  
 to the violation of the constraint. Thus, the threshold of the mapping assumes the 327  
 value of the distance between these attributes. 328

Attributes that were grouped in the mapping before the threshold update were 329  
 included by using an inappropriate threshold. Then, the mapping is dropped, and 330  
 its attributes are merged in new mappings around the two conflicting attributes. 331  
 Attributes from the original mapping that cannot be included in the newly generated 332  
 mappings (because their distance is greater than the new threshold) give rise to new 333  
 mappings. The thresholds of all the mappings created in these steps assume the 334  
 value of the distance between the attributes that have triggered the procedure. 335

To illustrate *SplitAndMerge*, consider the example shown in Fig. 13.4:  $m$  is a 336  
 mapping that groups the attributes  $\{a, b, c, e_1\}$ , with threshold  $t_m = 0.5$ , and  $e_2$  337  
 is an attribute that belongs to the same source of  $e_1$ . Suppose that the distance 338



**Fig. 13.4** *SplitAndMerge* over a mapping  $m$ . Labels on edges indicate distances:  $e_1$  and  $e_2$  belong to the same source, their distance is 0.29

between  $e_2$  and  $m$  is 0.3: it is lower than the mapping threshold 339  
 be added to  $m$ . However,  $m$  already contains  $e_1$ , which comes from the same 340  
 source of  $e_2$ , and thus violates the constraint. As a consequence, *SplitAndMerge* 341  
 removes the initial mapping  $m$ , and creates two new mappings,  $m'$  and  $m''$ . Each 342  
 of them is initialized with one of the conflicting attributes ( $e_1$  and  $e_2$ , in the 343  
 example); the threshold of these mappings is set to the value of the distance 344  
 between the two conflicting attributes (0.29). The other attributes of the initial 345  
 mapping are then grouped in the newly generated mapping, or they originate 346  
 new mappings. In the example, assuming that  $a$  and  $b$  have lower distances 347  
 from  $m'$  and  $m''$ , respectively, and that  $c$  has a distance greater than the new 348  
 threshold from both  $m'$  and  $m''$ , *SplitAndMerge* would generate the mappings 349  
 $m' = \{e_1, a\}$ ,  $m'' = \{e_2, b\}$ ,  $m''' = \{c\}$ , whose thresholds assume the value of the 350  
 distance between  $e_1$  and  $e_2$ . 351

Note that the effects of the approach propagate for all the remaining iterations; 352  
 in particular, observe that the distance between the attributes that triggered the split 353  
 is assigned to the thresholds of all the mappings generated by the procedure. 354

### 13.3.2 Wrapper Refinement

355

Once mappings are generated, we proceed to refine the wrappers. The key idea 356  
 behind the wrapper refinement process is that correct rules extract consistent 357  
 information from different sources; conversely, the values returned by weak rule 358  
 only partially overlap with those of other sources. Therefore, a weak rule extracts 359  
 inconsistent data for some tuples, producing high but not negligible distances with 360  
 the available mappings. 361

The refinement process corrects the wrapper by replacing a weak rule with an 362  
 alternative one that extracts consistent values, thus reducing the distance from the 363  
 mapping. A rule is considered weak if it produces attributes whose distance from the 364  
 mapping is greater than the average distance. A weak rule usually works correctly 365  
 for some pages (i.e., it extracts values that match with those extracted from other 366

sources), but fails with other pages (the extracted values do not match with other 367 sources). To refine the wrapper, we generate alternative rules that increase the 368 number of matching values, minimizing the distance of the corresponding attribute 369 from the mapping. 370

### 13.3.2.1 Extracting Labels

371

After all the sources have been processed (i.e., wrappers have been refined and 372 the final mappings have been computed), in a last step, we determine the labels 373 of the mappings. For each attribute belonging to a mapping, we process the pages 374 of the extracted values looking for meaningful semantic labels. We leverage the 375 observation that the labels of many attributes are part of the page template and occur 376 close to the values. 377

Our technique returns as candidate labels the textual template nodes that occur 378 closest to the extracted values. This technique may perform poorly on a single 379 source, but it leverages the redundancy among a number of Web sites to select 380 the best candidate labels. In other words, it is very unlikely that the same wrong 381 label is associated with a mapping as frequently as a meaningful one. Therefore, we 382 associate each mapping with all the labels associated with the corresponding rules, 383 and then we rank the labels according to the number of extraction rules associated 384 with them (higher is the number of rules, higher is the rank). 385

## 13.4 Assessing the Quality of Data

386

We have seen that according to our hidden relation model, Web sources provide 387 values for the attributes of a large number of objects. For example, financial 388 Web sites publish the values for several attributes, such as volume, open, 389 max and min values, etc. Unfortunately, different sources can report inconsistent 390 attribute values for the same object, making data published on the Web inherently 391 uncertain. 392

The uncertainty of data can be characterized by probability distribution functions: data are associated with functions reporting the probabilities that an attribute 393 assumes certain values for a given object. 394

This section describes state-of-the-art models to characterize the uncertainty 396 of Web data. These models have a twofold goal. They aim at computing (1) the 397 probability distributions for the data provided by the sources, and (2) the accuracy 398 of the sources with respect to the data of interest, that is, the probability that a 399 source provides the correct values for a set of objects. Three main factors influence 400 the development of these models: 401

Object	A: authority	I: independent	IC: ind. copied	C1: copier 1	C2: copier 2	NAIVE	ACCU	DEP
	Sources							
obj1	a	c	b	b	b	b	a—b	a
obj2	b	b	c	c	c	c	b	b
obj3	c	b	c	c	c	c	c	c
accuracy	1	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$			

**Fig. 13.5** Running example: *authority* always reports the correct value; *independent* and *independent copied* provide, independently from the other sources, one correct value out of three; *copier 1* and *copier 2* copy their values from *independent copied*

- *Consensus*: given an object, the larger is the number of sources that agree for the same value, the higher is the probability that the value is correct.
- *Accuracy*: the agreement of the sources' observations contributes in raising the probability that a value is correct in a measure that also depends on the accuracy of the involved sources.
- *Copiers*: the presence of copiers, that is, sources that publish data copied by other sources, can generate misleading consensus on the values proposed by the most copied sources.

As an example, consider five sources publishing the values of an attribute for the same three objects, as shown in Fig. 13.5. The first source A is an *authority* and provides the true value for each object (a, b, and c, respectively), while all the other sources (I, IC, C1, C2) provide a correct value only for one object out of three. The sources I and IC (*independent* and *independent copied*) provide their values independently from the other sources, while the remaining sources C1 and C2 merely copy the values proposed by IC. Notice that in normal settings, the role of the sources is not available as input to the models (i.e., they are not aware that A is an authority, I is independent, and so on).

### 13.4.1 Models Based on Consensus and Accuracy

419

The simplest model, called NAIVE,<sup>2</sup> for estimating the accuracy of sources and computing data distribution probabilities considers only the consensus: the most probable value is the one published by the largest number of sources, and the

<sup>2</sup>The names of the models presented in this chapter are inspired by those introduced by Dong et al. in [31].

probability of a value is estimated by its frequency over the given set of sources. 423  
 According to NAIVE, in our running example there are two possible values for obj2 424  
 provided by the five sources considered: b is provided by sources A and I, while b is 425  
 provided by three sources (IC and its copiers C1, C2). Similarly for obj1, the most 426  
 likely value would erroneously be the value b. The example shows how in presence 427  
 of many sources with different accuracies, a naive voting Forward could lead to 428  
 incorrect conclusions. The probability distribution of the NAIVE model corresponds 429  
 to the frequencies of the values. In the running example, the probability for obj1 is 430  
 $a \rightarrow 1/5, b \rightarrow 3/5, c \rightarrow 1/5.$  431

A more involved model, called ACCU, considers at the same time the first two 432  
 factors, consensus and accuracies, and produces as output an estimation of the 433  
 source accuracies together with the probabilities of the values [35, 45, 46]. Indeed, 434  
 consensus among sources and sources' accuracy is mutually dependent; the greater 435  
 is the accuracy of the sources, the more they agree for a large number of objects 436  
 and the more they will affect the general consensus. Similarly, the more the sources 437  
 agree on a large number of objects, the greater is their accuracy. 438

The role of the accuracies consists in weighting the consensus of the sources. 439  
 A voting Forward similar to that used with the NAIVE model can be used to estimate 440  
 the probabilities by means of the consensus: the only difference is that the votes are 441  
 weighted according to the accuracies of the sources. The accuracy of a source can 442  
 be estimated by comparing its observations with those of other sources for a set of 443  
 objects. A source that frequently agrees with other sources is likely to be accurate, 444  
 and similarly, the most accurate sources will be given the higher weights during the 445  
 computation of the probabilities of the true values. In our running example, consider 446  
 the accuracies given at the bottom of the table in Fig. 13.5: three sources (IC, C1, 447  
 and C2) provide the wrong value c for obj2, and they will be given an overall 448  
 accuracy of 1, while two sources (A,I) provide the correct value b with an overall 449  
 accuracy of  $\frac{4}{3}$ . However, even if the accuracies are known, the model still cannot 450  
 decide which value, between a and b, is the most likely value for obj1. 451

### 13.4.2 Dealing with Copying Sources

A more complex model also considers the presence of copiers, that is, sources 453  
 that publish values copied by one or more other sources. The presence of copiers 454  
 makes harder the problem of computing the true values and the accuracies of the 455  
 sources since they can create "artificial" consensus on values. A copier, even in good 456  
 faith, can propagate a wrong value originated in one of the sources from which it 457  
 copies. Provided that there is enough evidence about which are the correct values, 458  
 it is possible to detect which sources are copying, observing that copiers publish 459  
 the same false values of the sources from which they copy. For instance, if b is 460  
 considered the most likely value for obj2, the fact the IC, C1, and C2 publish the 461  
 same false value attests that there are two copiers. The same argument cannot be 462  
 used for obj3, for which the three sources publish the same value c: since this is a 463  
 true value, it is not necessarily an evidence of copying. 464

DEP is a model that considers all the three factors above: consensus, accuracy, 465 and copiers [31]. It tries to detect possible copiers by analyzing the dependencies 466 among the sources. Once the copiers has been detected, the consensus created 467 by their presence will be ignored during the computation of the probabilities. 468 The dependence analysis has to consider the mutual feedbacks among consensus, 469 accuracy, and dependencies: the accuracy of a source depends on the consensus over 470 the values it provides; the dependencies between sources depend on source accuracy 471 and the consensus over the values they provide; finally, the consensus should take 472 into account both the accuracy of sources and the dependencies between sources. 473 For instance, once that it has been detected that IC, C1, and C2 copy one another, 474 the voting expressed by two sources will be ignored, and then it can be established 475 that the most likely true value of obj1 is a. 476

In general, identifying the copiers is a challenging task for two main reasons. 477 First, if in the considered sources there is a lack of evidence, copiers can be missed. 478 Second, if the available evidence is misleading, false copiers can be detected. 479 In Fig. 13.6, we make use of an example to illustrate these issues: four distinct 480 Web sources report data about the same three soccer players. For each player, 481 three attributes are reported: *position*, *height*, and *birth date*. The fifth table shows 482 the true values for the considered scenario. We remark that such information is 483 not provided in general; in this example, we consider it as given to facilitate the 484 discussion. 485

Consider now the first attribute, the position of the player. It is easy to notice 486 that Web source 1 and Web source 2 are reporting the same false value for the 487 position of Messi (errors are in bold). Following the intuition from [31], according 488 to which copiers can be detected as the sources share false values, they should be 489 considered as copiers. Conversely, observe that Web source 3 and Web source 4 490 report only true values for the position, and therefore, there is not any significant 491 evidence of dependence. The scenario radically changes if we look at the other 492 attributes. Web source 3 and Web source 4 are reporting the same incorrect 493 values for the birth date attribute, and they also make a common error for the 494 height attribute. Web source 4 also reports independently an incorrect value for 495 the height of Totti. In this scenario our approach concludes that Web source 3 496 and Web source 4 are certainly dependent, while the dependency between Web 497 source 1 and Web source 2 would be very low. Using the DEP model, therefore 498 by looking only to a single attribute at the time, Web source 1 and Web source 2 499 would be reported as copiers for the position attribute because they share the same 500 formatting rule for such data (i.e., false copiers detected), while Web source 3 501 and Web source 4 would be considered independent sources (i.e., real copiers 502 missed). 503

Starting from the above observations, the dependence analysis has been further 504 investigated and a more complex model M-DEP has been introduced to consider not 505 only single attributes at a time, but whole tuples [12, 30]. 506

AQ5

**Web source 1**

	<i>Position</i>	<i>Height</i>	<i>Birth date</i>
Francesco Totti	Attacking midfielder	180	27 September 1976
Lionel Messi	<b>Striker</b>	169	24 June 1987
Wayne Rooney	Striker	176	24 October 1985

**Web source 2**

	<i>Position</i>	<i>Height</i>	<i>Birth date</i>
Francesco Totti	Attacking midfielder	180	27 September 1976
Lionel Messi	<b>Striker</b>	169	24 June 1987
Wayne Rooney	Striker	176	24 October 1985

**Web source 3**

	<i>Position</i>	<i>Height</i>	<i>Birth date</i>
Francesco Totti	Attacking midfielder	180	27 September 1976
Lionel Messi	Forward	<b>170</b>	<b>June 1987</b>
Wayne Rooney	Striker	176	<b>October 1985</b>

**Web source 4**

	<i>Position</i>	<i>Height</i>	<i>Birth date</i>
Francesco Totti	Attacking midfielder	<b>181</b>	27 September 1976
Lionel Messi	Forward	<b>170</b>	<b>June 1987</b>
Wayne Rooney	Striker	176	<b>October 1985</b>

*True values*

	<i>Position</i>	<i>Height</i>	<i>Birth date</i>
Francesco Totti	Attacking midfielder	180	27 September 1976
Lionel Messi	Forward	169	24 June 1987
Wayne Rooney	Striker	176	24 October 1985

**Fig. 13.6** Four Web sources reporting data for three soccer players. The last table represents the true values for the scenario

## 13.5 Experiments

507

We now present some experimental results for the three main topics discussed in the chapter. We first discuss the effectiveness of the proposed solution to automatically gather sources of interest. Then we present the results obtained by the algorithms for the extraction and integration of data from these sources. Finally, we show how current state-of-the-art models for the reconciliation of conflicting values deal with such real-world Web data.

508

509

510

511

512

513

### 13.5.1 Discovering New Sources

514

We developed OUTDESIT [11], a prototype that implements the techniques described in Sect. 13.2, and we used it to perform experiments to validate the

515

516

proposed techniques. Here, we focus on the results that we obtained in the *soccer* 517 domain, as published information is easy to interpret and there are no clear 518 authorities such as in other domains. Namely, we report some results on our 519 experiment aimed at discovering new sources with pages publishing data about 520 instances of the SOCCERPLAYER conceptual entity. 521

AQ6

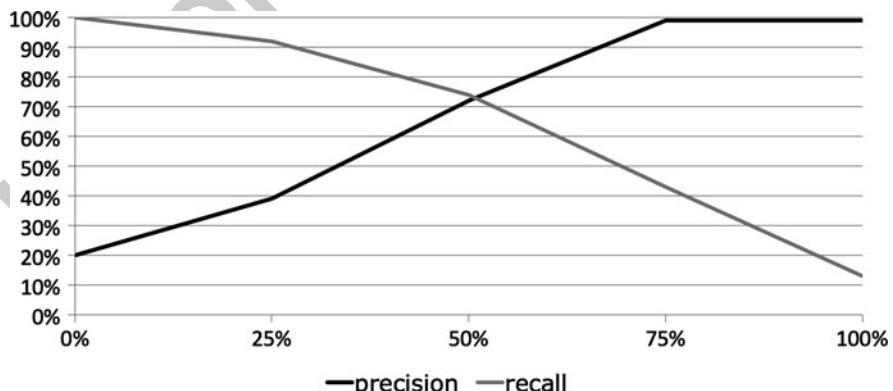
### 13.5.1.1 Entity Descriptions

522

We have taken three sample pages, from three different Web sites, each one 523 publishing data about one player and we have run OUTDESIT on it. The generated 524 entity description contains the attributes: *club*, *height*, *nationality*, and *weight*, while 525 the keyword extracted from the sample pages is *soccer*. 526

We manually analyzed the behavior of the filtering function in the search 527 expansion step, that is, the use of the entity description to check whether a given 528 page is valid for our purposes. We have run a single iteration of OUTDESIT with a 529 set of identifiers pointing to 500 SOCCERPLAYER pages, selected randomly from 530 ten soccer Web sites. The search engine returned about 15,000 pages distributed 531 over 4,000 distinct Web sites. We then manually evaluated the Web sites to measure 532 the precision and the recall of the function over such pages. In particular, we studied 533 how precision and recall behave by varying the percentage of description keywords 534 that are required in the template to accept a page (100% means that all the keywords 535 are needed). 536

Figure 13.7 shows how by raising the threshold, the precision increases and the 537 recall decreases. The system achieves 100% precision when the number of keywords 538 from the description required to be in the template of the page is at least 75%. When 539 only 50% of the keywords are required, the accepted pages are 74% of the total 540 valid pages returned by the search engine, and the precision is 72%. Examples of 541



**Fig. 13.7** Performance of the filtering function varying the threshold  $t$

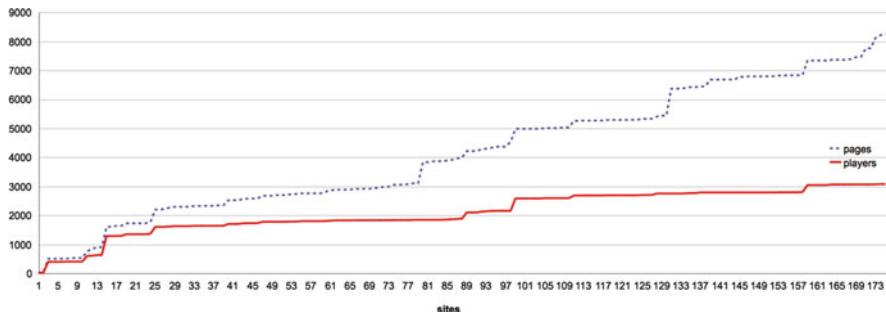


Fig. 13.8 Pages and players found by OUTDESIT

nonvalid pages returned by the search engine results are personal pages and blogs, news, and forum pages: they are pertinent with the keywords in the queries, but they are not instance of the entity as in our definition. As these terms did not appear in the pages template, our function correctly discarded them.

### 13.5.1.2 Quantitative Evaluation

546

Figure 13.8 depicts the number of pages discovered by OUTDESIT for the target entity. The graph plots the number of new instance pages against the number of new Web sites discovered. Starting from three sample pages, for each entity the algorithm automatically discovers several thousands of pages. By a manual inspection, conducted on a representative subset of the results, we can conclude that all the retrieved pages can be considered as instances of the entity exemplified by the input sample pages. The graphs also plot the number of distinct anchors (used as identifiers) that are found in each step. Somehow they can approximate the number of distinct players. As expected, it is evident that they increase less than the number of pages.

556

### 13.5.2 Extracting and Integrating Web Data

557

To evaluate the approach described in Sect. 13.3, we implemented in a prototype the data extraction and integration algorithms [7, 8] and run them with some collections of pages. Here we report the results concerning the *soccer* and *finance* domains. For each domain, we let OUTDESIT collect 100 sources. Each source consists of tens to thousands of pages, and each page contains detailed data about one object of the corresponding entity.

563

We used the standard metrics precision, recall, and F-measure to compare hand-crafted mappings with those automatically produced by the prototype: for each mapping  $A$  generated by the prototype with respect to a golden mapping  $B$ , we computed:  $P = \frac{|A \cap B|}{|A|}$ ;  $R = \frac{|A \cap B|}{|B|}$ ;  $F = \frac{2 * P * R}{P + R}$ .

567

### 13.5.2.1 Qualitative Evaluation

568

To evaluate the quality of the mappings, for each domain we selected 20 Web sources (the largest ones) and we manually built a golden set of mappings by inspecting ten pages per source; only the attributes that were present in at least three sources were included in the mappings. The golden schema of the stock quote entity contains 29 attributes, while that of the soccer players contains 14.

For each domain, we ran four executions to evaluate the impact of the Split-AndMerge and of the wrapper refinement on the quality of the inferred mappings. As expected, the best performances in terms of precision and F-measure are always obtained when both the *SplitAndMerge* and the wrapper refinement were activated. We report that the *NaiveMatch* (i.e., simple match without the *SplitAndMerge* technique) alone always obtains mappings with high recall but with low precision, especially in the finance domain. In fact, *NaiveMatch* gathers many valid attributes, but it aggregates several heterogeneous attributes within the same mapping, as it is not able to distinguish attributes with similar values, thus producing many false positives. The precision of the *SplitAndMerge* algorithm greatly benefits of the more advanced matching technique, especially in the finance domain. It is interesting to observe the direct correlation between the thresholds that have been dynamically increased and the improvements in the results. In the *finance* domain, which contains several similar values, an average increment of 37% for the threshold corresponds to an improvement of the precision of 250%. In the *soccer* domain, an increment of 32% for the threshold corresponds to a precision gain of 82%.

The wrapper refinement has always positive impacts on the performance since it increases both precision and recall: as extraction rules are improved, some attributes can reach a sufficient matching score to be added in the mapping set.

### 13.5.2.2 Quantitative Evaluation

593

Table 13.1 reports the execution of the system against the whole sets of data sources. We report for each of the eight largest output mappings, the mapping cardinality ( $|mapping|$  (i.e., the number of extraction rules in each of them) and the most likely label inferred by the system.

According to the model introduced in Sect. 13.1, each source provides only a partial view of the hidden relation. The mappings and the extraction rules produced by our system allow us to build an integrated view of its whole extension by accumulating information from many sources, thus obtaining more information than actually exposed by each participating source. To give an example, consider the objects and the eight attributes reported for the soccer and the finance domains in Table 13.1: the hidden relations for these entities contain at most 224 k values (28 k objects  $\times$  8 attributes) and 4.6 k ( $5.76 \times 8$ ) values, respectively. In the finance domain, a single source covers on average about 3.2 k values (70% of the total), while the integrated view over the 100 sources reaches 4.1 k values (90% of the

**Table 13.1** Top-8 results for 100 Web sources: for each mapping its most likely label and its cardinality are reported

Soccer players		Stock quotes		
	45,714 pages (28,064 players)		56,904 pages (576 stock quotes)	
Label	Mapping	Label	Mapping	t55.1
Name	90	Stock	84	t55.2
Birth	61	Price	73	t55.3
Height	54	Change	73	t55.4
Nationality	48	Volume	52	t55.5
Club	43	Low	43	t55.6
Position	43	High	41	t55.7
Weight	34	Last	29	t55.8
League	14	Open	24	t55.9
				t55.10

total). More interestingly, in the soccer domain, a single source covers on average 608 for only 1.4 k values (1% of the total), while the integrated view reaches 134 k values 609 (71%). As for the same object and attribute different values are provided by distinct 610 sources, conflicting values can arise. We discuss next the experimental results for 611 recent works that address this issue. 612

### 13.5.3 Assessing the Quality of Data

613

We implemented the models described in Sect. 13.4 to validate experimentally the 614 proposed techniques on the data provided by real-life Web data sources. Here we 615 concentrate on the results obtained in the soccer and finance domains. In particular, 616 we report the following data extracted and integrated by our algorithms: Height, 617 Weight, and BirthDate for soccer players; Price, Open Value, and Volume for 618 stock quotes. For these attributes, we manually produced the correct (true) values 619 for a set of 861 stock quotes and 100 soccer players. Stock quotes and players 620 were selected randomly, making sure that both popular and rare objects were part of 621 the set. 622

For the stock quote domain, the true values were collected by means of an 623 authoritative source (the Nasdaq official Web site). For soccer players, since an 624 authoritative source does not exist, the true values of the considered attributes 625 were manually collected by inspecting the official Web site of every soccer 626 player, whenever available, and the official Web site of his current team club, 627 otherwise. 628

### 13.5.3.1 Accuracy of Web Data Sources

629

Given a truth vector  $T = [t_1, \dots, t_n]$  of correct values for an attribute, we define the sampled accuracy as the fraction of true values correctly reported by the site over the number of objects in the truth vector for which it publishes a value. For example, suppose we want to compute the sampled accuracy of a soccer Web site  $w$  reporting Height values for 1,000 objects. We match this set of objects with the true values for Height in  $T$  and identify 80 soccer players in the intersection of the two sets. We can now compute the sampled accuracy  $\bar{a}_i$  for the source  $i$ : if, for example, the values reported by the source coincide with values in  $T$  for 40 objects, then we estimate that the source reports true values for 50% of the cases, and therefore,  $\bar{a}_i=0.5$ . We compute in a similar way the sampled accuracy  $a_i^m$  for every evaluated model  $m$ , the only difference is that the set of values matched with  $T$  is the one made by the most probable values computed by  $m$ .

We computed the sampled accuracy of the sources for the Height (soccer players) and the Open Value (stock quotes) attributes. The average source accuracy is 70.21% for the first and 85.78% for the latter. Sampled source accuracy is better for domains where at least an authority exists. In fact, Open Value exhibits high source accuracy (more than 78%) for every source, while in the case of the soccer players' Height and Weight, the source accuracies are sensibly lower in all the sources. Overall, results show that attributes from distinct domains may assume quite different behaviors. Conversely, the sampled accuracies of the other attributes in the same domain behave quite similarly.

### 13.5.3.2 Applying Different Probabilistic Models

651

Figure 13.9 reports how the probabilistic models NAIVE, ACCU, DEP, and M-DEP perform by using the three metrics introduced before. *Sampled accuracy* shows that, on average, all models are able to identify the correct values with high precision in most cases. As expected, more complex models present better results for all domains, but the best model, M-DEP, outperforms, the simplest one, NAIVE, only by 3.5% on average.

The *probability concentration* measures how much probability a model has concentrated on the known true value; i.e., the lower is  $PC$ , the more the probability distributions are scattered over incorrect values. We assume that the NAIVE model proposes probability distribution functions that merely reflect the frequencies of the observed values. This metrics shows that, on average, the most complex models ACCU, DEP, and M-DEP constantly outperform NAIVE by 14.2%, 15.5%, and 18.2%, respectively, as tools for estimating probability distribution functions. We notice that the most sensible differences are in the domain where an authoritative source exists.

Finally, the *accuracy distance* metrics measure the distance between the source quality estimation made by a model and their real sampled accuracy. On average, NAIVE is only marginally outperformed by the other models in estimating the

	Sampled accuracy			Probability concentration			Accuracy distance					
	NAIVE	ACCU	DEP	M-DEP	NAIVE	ACCU	DEP	M-DEP	NAIVE	ACCU	DEP	M-DEP
<b>Birth date</b>	0.98	0.97	0.98	0.98	0.82	0.97	1.00	1.00	8.58	2.28	2.28	2.72
<b>Height</b>	0.66	0.67	0.67	0.67	0.51	0.66	0.67	0.67	4.6	13.86	13.93	16.49
<b>Weight</b>	0.59	0.66	0.66	0.66	0.49	0.67	0.67	0.67	6.37	10.53	10.76	12.11
<b>Price</b>	0.96	0.95	0.96	0.96	0.95	0.95	0.95	1	2.1	2.3	3.18	1.58
<b>Open value</b>	0.97	0.95	0.97	0.97	0.73	0.95	0.96	0.96	11.01	9.56	9.74	8.39
<b>Volume</b>	1	1	1	1	0.83	1	1	1	13.12	0	0	0
<b>Avg</b>	0.88	0.88	0.89	0.91	0.77	0.88	0.89	0.91	6.74	6.16	6.28	5.29

**Fig. 13.9** The sampled accuracy (higher is better), the probability concentration (higher is better), and the accuracy distance (lower is better)

correct accuracies of the input sources. This behavior is due to the different characteristic of the observed data. In the stock quotes scenario, there is a large number of distinct symbols in the alphabet and a larger number of instances. Also Birthdate has a larger alphabet than Height or Weight. We refer the reader to [12] for more details on the role of the alphabet.

In general, for applications requiring only an estimation of the true values, even a simple approach such as the NAIVE model can be used. However, whenever a more precise characterization of the uncertainty is required, for example, to populate a probabilistic database [9], complex models exhibit significant advantages.

## 13.6 Related Work

679

The scientific literature is presented by grouping the related works in five categories related to our approach: the work related to the discovery of sources is described in Sect. 13.6.1; the data integration and information extraction related works are discussed in Sects. 13.6.2 and 13.6.3, respectively; Sect. 13.6.4 contains a description of related work in the wrapper generation field, and finally, Sect. 13.6.5 concludes this discussion considering works related to the analysis of the quality of data and sources on the Web.

686

### 13.6.1 Source Discovery

687

Our work is related to researches on focused crawlers (or topical crawlers) [20, 39, 43], which face the issue of efficiently fetching Web pages that are relevant to a specific topic. Focused crawlers typically rely on text classifiers to determine the relevance of the visited pages to the target topic. Page relevance and contextual information—such as, the contents around the link, the lexical content of ancestor pages—are used to estimate the benefit of following URLs contained in the most of relevant pages. Although focused crawlers present some analogy with our work, our goal is different as we aim at retrieving pages that publish the same type of information, namely, pages containing data that represent instances of the entity exemplified by means of a few sample pages.

697

Vidal et al. present a system, called GOGETIT! that takes as input a sample page and an entry point to a Web site and generates a sequence of *URL patterns* for the links a crawler has to follow to reach pages that are structurally similar to the input sample [44]; therefore, their approach can be used as an implementation of the step of our crawling strategy that collects within a Web site all the pages similar to an input page.

703

### 13.6.2 Information Extraction

704

DIPRE [16] represents a pioneering technique to extract relations from the Web. 705 Starting from a bunch of seed pairs (e.g., author-book), it collects similar pairs 706 by means of a process in which the research of new pages containing these pairs 707 and the inference of patterns extracting them are interleaved. The applicability of 708 the approach is limited, since it cannot deal with generic tuples of more than two 709 attributes, but the paper motivated several Web information extraction projects to 710 develop effective techniques for the extraction of facts (binary predicates, e.g., born- 711 in⟨scientist, city⟩) from large corpora of Web pages (e.g., [1, 4]). 712

Web information extraction techniques mainly infer lexico-syntactic patterns 713 from textual descriptions, but they cannot take advantage of the structures available 714 on the Web, as they do not elaborate data that are embedded in HTML templates. 715 For instance, Cafarella et al. developed a system to populate a probabilistic database 716 with data extracted from the Web [17], but the data extraction task is performed 717 by TEXTRUNNER [4], an information extraction system which is not suitable for 718 working on data-rich Web pages that are the target of our searches. 719

Compared to our system, these approaches are not able to exploit the information 720 offered by data-rich pages. In fact, they concentrate on the extraction of facts: large 721 collections of named entities (e.g., names of scientists, politicians, cities) or simple 722 binary predicates. Moreover, they are effective with facts that appear in well-phrased 723 sentences, whereas they fail to elaborate data that are implied by Web page layout 724 or markup practices, such as those typically published in Web sites containing data- 725 rich pages. 726

### 13.6.3 Data Integration

727

The bootstrap of data integration architecture for structured data on the Web is 728 the subject of [40]. However, the presented system, PAYGO, focuses on explicitly 729 structured sources, such as Google Base, and the proposed integration techniques 730 are based on the availability of attribute labels; on the contrary, our approach 731 aims at integrating unlabeled data from Web sites and automatically infers the labels 732 whenever they are encoded in the HTML templates. 733

The exploitation of structured Web data is the primary goal of WebTables [19], 734 which concentrates on data published in HTML tables. Compared to informa- 735 tion extraction approaches, WebTables extracts relations with involved relational 736 schemas but it does not address the issue of integrating the extracted data. 737

Another research project that addresses issues related to ours is CIMPLE [41] 738 whose goal is to develop a platform to support the information needs of the members 739 of a virtual community [29]. Compared to our method, CIMPLE requires an expert 740 to provide a set of relevant sources and to design an Entity-Relationship model 741 describing the domain of interest. Both CIMPLE and OCTOPUS [18] systems support 742

users in the creation of datasets from Web sites by means of a set of operators to 743  
perform search, extraction, data cleaning, and integration. Although such systems 744  
have a more general application scope than ours, they involve the user in the process, 745  
while our approach is completely automatic. 746

Similarly, TAP and SEMTAG by Guha et al. [26, 36] are related projects that 747  
require human intervention: TAP involves knowledge extracted from structured Web 748  
pages and encoded as entities, attributes, and relations; SEMTAG provides a semantic 749  
search capability driven by the TAP knowledge base. Contrary to our approach, TAP 750  
requires hand-crafted rules for each site that it crawls, and when the formats of those 751  
sites change, the rules need to be updated. 752

Also, the MetaQuerier system developed by Chang et al. has similar objectives 753  
to our proposal, as it aims at supporting exploration and integration of databases 754  
on the Web [21]. However, this approach and its methods only concentrate on the 755  
deep-web. 756

The idea of using duplicate instances in the matching process to deal with 757  
imprecise data and schemas has been recently studied (e.g., [6]): these proposals 758  
show how the redundancy can help in contexts where schema can be imprecise. 759  
However, these approaches are not conceived to deal with the Web scale. 760

Data instances and domain constraints are used also in Glue [28], which early 761  
introduced a framework for finding semantic mappings between concepts of ontolo- 762  
gies. Although the Glue approach has a general applicability in the semantic Web 763  
context, it is not suitable in our setting, since it relies also on intensional aspects, 764  
such as element names of the ontology taxonomy and the hierarchical relationships 765  
among elements, while we make a stronger exploitation of the redundancy of 766  
information that occur at the extensional level. 767

Many works try to solve heterogeneity problems in duplicate detection, by 768  
composing different matching techniques [42] to deal with imprecise information, 769  
or by taking advantage from data structure, to avoid ambiguity [27]. In the schema- 770  
matching literature, there is not known an approach that considers duplicates with 771  
several kinds of errors (e.g., misspelling, misplaced characters), while in the record- 772  
linkage literature, most of the approaches developed need intensional descriptions 773  
to work. 774

### 13.6.4 Wrapper Inference and Refinement

775

An important issue in our solution is the automatic generation of wrappers. 776  
RoadRunner [24] and ExAlg [3] propose automatic inference techniques to create 777  
a wrapper from a collection of pages generated by the same template. These 778  
approaches are not directly suitable to our goals: they do not consider effective 779  
techniques to scale with the number of sources, and they generate complex nested 780  
schemas that can be hard to integrate when dealing with a large number of sources. 781  
However, the presented approach can be considered to further improve the level 782

of automation these unsupervised techniques exhibit in a slightly different setting 783 involving multiple input Web sources [14]. 784

An approach more closely related to ours is developed in the TurboWrapper 785 project [22], which introduces a composite architecture including several wrapper 786 inference systems (e.g., [3, 24]). By means of an integration step of their output, 787 it aims at improving the results of the single participating systems taken separately. 788 Interestingly, the design of TurboWrapper is motivated by the observation that 789 different Web sources that offer data for the same domain have a strong redundancy 790 at the schema level. However, compared to our solution, it makes stronger domain- 791 dependent assumptions for the integration step since it does not consider the 792 redundancy of information that occurs at the instance level: the correlation of data 793 in the integration step is based on the assumption that there exist distinguishable 794 generative models for the attributes to be matched (e.g., the *isbn* in the book 795 domain). This is a strong domain-dependent assumption that limits the application 796 of the technique, since in several domains, such as the finance domain, distinct 797 attributes might have a too similar underlying generative model, e.g., min and max 798 price of a stock quote. 799

Our wrapper refinement phase resembles the intuitions behind the “augment 800 method” in [37], with the remarkable difference that we automatically gather the 801 corpus during the integration process, while in their case, the corpus is given as 802 input. In fact, a direct application of their approach is not possible in our setting, 803 since we do not consider a priori information about the domain (i.e., at bootstrap we 804 do not have any corpus of schemas nor mappings). 805

### 13.6.5 Assessing the Quality of Data

806

Our work explores the application of probabilistic techniques to assign truth 807 probabilities to values gathered from conflicting sources of information. Such 808 information is then used to evaluate the quality of the Web sources in terms of 809 their data accuracy. The problem is related to the broader context of data quality [5] 810 and to the issue of combining probability distributions expressed by a group of 811 experts, which has been studied in the statistics community (e.g., [23]). A related 812 experimental comparison of authority and quality results for Web sites has been 813 done in [2]. 814

Many projects have recently been active in the study of imprecise databases and 815 have achieved a solid understanding of how to represent and process uncertain data 816 (see [25] for a survey on the topic). 817

The development of effective data integration solutions making use of probabilis- 818 tic approaches has also been addressed by several projects in the last years. In [33], 819 the redundancy between sources is exploited to gain knowledge, but with a different 820 goal: given a set of text documents, they assess the quality of the extraction process. 821 Other works propose probabilistic techniques to integrate data from overlapping 822 sources [34]. 823

On the contrary, only recently, there has been some focus on how to populate such databases with sound probabilistic data. Even if this problem is strongly application specific, there is a lack of solution. In [17], Cafarella et al. describe a system to populate a probabilistic database with data extracted from the Web, but they do not consider the problems of combining different probability distributions and evaluating the reliability of the sources.

TruthFinder [46] was the first project to address the problem of truth discovery in the presence of multiple Web sources providing conflicting information. TruthFinder considers both the consensus on values and the accuracy of sources, and it can be considered as the first work that realizes and exploits their mutual dependency. Based on some heuristics, an iterative algorithm computes the trustiness of values and the accuracy of the sources. A similar direction has been also explored by Wu et al. [45] and [35] which present fix-point algorithms to estimate the true value of data reported by a set of sources, together with the accuracy of the sources.

Some of the intuitions behind TruthFinder were formalized in a probabilistic Bayesian framework by Dong et al. [31], who also considered how the presence of copiers (i.e., sources that copy from other sources) affects the evaluation of the source accuracy. While in TruthFinder the effects of possible copying dependencies between sources are handled by means of a simple heuristic, the authors of [31] develop a more principled approach to detect source dependencies. To achieve these results, their model (which corresponds to the DEP model illustrated in Sect. 13.4) computes the probability that a pair of sources are dependent by analyzing the co-occurrences of errors. A further variant by the same authors also considers the variations of truth values over time [32].

The model behind DEP has been extended to improve the quality of the source dependencies detection. In fact, in [31], sources are seen as providers that supply data about a collection of objects, i.e., instances of a real-world entity, such as a collection of video games. However, it is assumed that objects are described by just one attribute, e.g., the publisher of a video game. On the contrary, data sources usually provide complex data, i.e., collections of tuples with many attributes, and it has been shown that by considering this information, the quality of the results can be further improved [12, 30].

## 13.7 Conclusions

This chapter described methods and techniques to wring out value from Web data. We concentrate on data-intensive Web sites that publish large amounts of data. Our approach builds on an abstract model that describes the generation of pages offered by these sites. The model emphasizes two important features that inspired our proposal: the redundancy of information that occurs on the Web and the presence of local regularities exhibited by data-intensive Web sites.

We described methods to discover and collect these pages. Given as input only 864  
a small set of sample pages, our method creates a description of the target entity, 865  
interacts with a search engine to locate new sources, and filters out the Web sites 866  
that do not fit the inferred description. 867

Then we introduced an approach to extract and integrate data from these sources. 868  
Our approach introduces an instance-based schema matching algorithm, which is 869  
adaptive to the actual data, thus presenting significant advantages in general settings 870  
where no a priori knowledge about the domain is given, and multiple sources have to 871  
be matched. Also the approach takes advantage of the coupling between the wrapper 872  
inference and the data integration tasks to improve the quality of the wrappers. 873

Since Web data are inherently imprecise, conflicting values usually occur among 874  
the data provided by the sources. Therefore, we presented state-of-the-art models 875  
to manage the uncertainty of Web data. These models compute the accuracy of the 876  
sources, and associate a probability distribution with the data. 877

To give a flavor of the effectiveness of the presented methods and techniques, 878  
we finally reported the results of some experiments conducted over real-world data 879  
from different domains. 880

## References

881

1. Agichtein, E., Gravano, L.: Snowball: extracting relations from large plain-text collections. DL 882  
'00, pp. 85–94 (2000) 883
2. Amento, B., Terveen, L.G., Hill, W.C.: Does “authority” mean quality? predicting expert 884  
quality ratings of web documents. SIGIR, pp. 296–303 (2000) 885
3. Arasu, A., Garcia-Molina, H.: Extracting structured data from web pages. ACM SIGMOD 886  
international conference on management of data (SIGMOD'2003), San Diego, California, 887  
pp. 337–348 (2003) 888
4. Banko, M., Cafarella, M., Soderland, S., Broadhead, M., Etzioni, O.: Open information 889  
extraction from the web. IJCAI (2007) 890
5. Batini, C., Scannapieco, M.: Data Quality: Concepts, Methodologies, and Techniques. 891  
Springer, Berlin, Heidelberg, New York (2008) 892
6. Bilke, A., Naumann, F.: Schema matching using duplicates. ICDE, pp. 69–80 (2005) 893
7. Blanco, L., Bronzi, M., Crescenzi, V., Merialdo, P., Papotti, P.: Exploiting information 894  
redundancy to wring out structured data from the web. In: Rappa, M., Jones, P., Freire, J., 895  
Chakrabarti, S. (eds.) WWW, pp. 1063–1064. ACM, New York (2010) 896
8. Blanco, L., Bronzi, M., Crescenzi, V., Merialdo, P., Papotti, P.: Redundancy-driven web data 897  
extraction and integration. WebDB (2010) 898
9. Blanco, L., Bronzi, M., Crescenzi, V., Merialdo, P., Papotti, P.: Automatically building 899  
probabilistic databases from the web. WWW (Companion Volume), pp. 185–188 (2011) 900
10. Blanco, L., Crescenzi, V., Merialdo, P.: Efficiently locating collections of web pages to wrap. 901  
WEBIST (2005) 902
11. Blanco, L., Crescenzi, V., Merialdo, P., Papotti, P.: Supporting the automatic construction of 903  
entity aware search engines. WIDM, pp. 149–156 (2008) 904
12. Blanco, L., Crescenzi, V., Merialdo, P., Papotti, P.: Probabilistic models to reconcile complex 905  
data from inaccurate data sources. CAiSE, pp. 83–97 (2010) 906
13. Blanco, L., Crescenzi, V., Merialdo, P., Papotti, P.: Contextual data extraction and instance- 907  
based integration. International workshop on searching and integrating new web data sources 908

(VLDS) (2011)	909
14. Blanco, L., Crescenzi, V., Merialdo, P., Papotti, P.: Wrapper generation for overlapping web sources. <i>Web Intelligence (WI)</i> (2011)	910 911
15. Blanco, L., Dalvi, N.N., Machanavajjhala, A.: Highly efficient algorithms for structural clustering of large websites. <i>WWW</i> , pp. 437–446 (2011)	912 913
16. Brin, S.: Extracting patterns and relations from the World Wide Web. <i>Proceedings of the First Workshop on the Web and Databases (WebDB'98)</i> (in conjunction with EDBT'98), pp. 102–108 (1998)	914 915 916
17. Cafarella, M.J., Etzioni, O., Suciu, D.: Structured queries over web text. <i>IEEE Data Eng. Bull.</i> <b>29</b> (4), 45–51 (2006)	917 918
18. Cafarella, M.J., Halevy, A.Y., Khoussainova, N.: Data integration for the relational web. <i>PVLDB</i> <b>2</b> (1), 1090–1101 (2009)	919 920
19. Cafarella, M.J., Halevy, A.Y., Wang, D.Z., Wu, E., Zhang, Y.: Webtables: exploring the power of tables on the web. <i>PVLDB</i> <b>1</b> (1), 538–549 (2008)	921 922
20. Chakrabarti, S., van den Berg, M., Dom, B.: Focused crawling: a new approach to topic-specific Web resource discovery. <i>Comput. Networks (Amsterdam, Netherlands)</i> <b>31</b> (11–16), 1623–1640 (1999)	923 924 925
21. Chang, K.C.C., Bin, H., Zhen, Z.: Toward large scale integration: building a metaquerier over databases on the web. <i>CIDR 2005</i> , pp. 44–66 (2005)	926 927
22. Chuang, S.L., Chang, K.C.C., Zhai, C.X.: Context-aware wrapping: synchronized data extraction. <i>VLDB</i> , pp. 699–710 (2007)	928 929
23. Clemen, R.T., Winkler, R.L.: Combining probability distributions from experts in risk analysis. <i>Risk Anal.</i> <b>19</b> (2), 187–203 (1999)	930 931
24. Crescenzi, V., Mecca, G., Merialdo, P.: ROADRUNNER: towards automatic data extraction from large Web sites. <i>International conference on very large data bases (VLDB 2001)</i> , Roma, Italy, 11–14 September 2001, pp. 109–118	932 933 934
25. Dalvi, N.N., Suciu, D.: Management of probabilistic data: foundations and challenges. <i>PODS</i> , pp. 1–12 (2007)	935 936
26. Dill, S., Eiron, N., Gibson, D., Gruhl, D., Guha, R., Jhingran, A., Kanungo, T., Rajagopalan, S., Tomkins, A., Tomlin, J.A., Zien, J.Y.: Semtag and seeker: bootstrapping the semantic web via automated semantic annotation. <i>WWW '03: proceedings of the 12th International Conference on World Wide Web</i> , pp. 178–186. ACM, New York, NY, USA (2003). <a href="http://doi.acm.org/10.1145/775152.775178">http://doi.acm.org/10.1145/775152.775178</a>	937 938 939 940 941
27. Do, H.H., Rahm, E.: Matching large schemas: approaches and evaluation. <i>Inf. Syst.</i> <b>32</b> (6), 857–885 (2007)	942 943
28. Doan, A., Madhavan, J., Domingos, P., Halevy, A.: Learning to map between ontologies on the semantic web. <i>WWW '02</i> , pp. 662–673 (2002)	944 945
29. Doan, A., Ramakrishnan, R., Chen, F., DeRose, P., Lee, Y., McCann, R., Sayyadian, M., Shen, W.: Community information management. <i>IEEE Data Eng. Bull.</i> <b>29</b> (1), 64–72 (2006)	946 947
30. Dong, X., Berti-Equille, L., Hu, Y., Srivastava, D.: Global detection of complex copying relationships between sources. <i>PVLDB</i> <b>3</b> (1), 1358–1369 (2010)	948 949
31. Dong, X.L., Berti-Equille, L., Srivastava, D.: Integrating conflicting data: the role of source dependence. <i>PVLDB</i> <b>2</b> (1), 550–561 (2009)	950 951
32. Dong, X.L., Berti-Equille, L., Srivastava, D.: Truth discovery and copying detection in a dynamic world. <i>PVLDB</i> <b>2</b> (1), 562–573 (2009)	952 953
33. Downey, D., Etzioni, O., Soderland, S.: A probabilistic model of redundancy in information extraction. <i>IJCAI</i> , pp. 1034–1041 (2005)	954 955
34. Florescu, D., Koller, D., Levy, A.Y.: Using probabilistic information in data integration. <i>VLDB</i> , pp. 216–225 (1997)	956 957
35. Galland, A., Abiteboul, S., Marian, A., Senellart, P.: Corroborating information from disagreeing views. <i>Proceedings of WSDM</i> , New York, USA (2010)	958 959
36. Guha, R., McCool, R.: Tap: a semantic web platform. <i>Comput. Networks</i> <b>42</b> (5), 557–577 (2003)	960 961

37. Madhavan, J., Bernstein, P.A., Doan, A., Halevy, A.Y.: Corpus-based schema matching. ICDE, pp. 57–68 (2005) 962  
963
38. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008). <http://www.informationretrieval.org> 964  
965
39. Pant, G., Srinivasan, P.: Learning to crawl: comparing classification schemes. ACM Trans. Inf. Syst. **23**(4), 430–462 (2005) 966  
967
40. Sarma, A.D., Dong, X., Halevy, A.Y.: Bootstrapping pay-as-you-go data integration systems. SIGMOD conference, pp. 861–874 (2008) 968  
969
41. Shen, W., DeRose, P., McCann, R., Doan, A., Ramakrishnan, R.: Toward best-effort information extraction. SIGMOD conference, pp. 1031–1042 (2008) 970  
971
42. Shen, W., DeRose, P., Vu, L., Doan, A., Ramakrishnan, R.: Source-aware entity matching: a compositional approach. ICDE, pp. 196–205. IEEE Computer Society, Silver Spring, MD (2007) 972  
973  
974
43. Sizov, S., Biwer, M., Graupmann, J., Siersdorfer, S., Theobald, M., Weikum, G., Zimmer, P.: The bingo! system for information portal generation and expert web search. CIDR 2003, First Biennial conference on innovative data systems research, Asilomar, CA, USA, 2003 975  
976  
977
44. Vidal, M.L.A., da Silva, A.S., de Moura, E.S., Cavalcanti, J.M.B.: Structure-driven crawler generation by example. In: Eftimidis, E.N., Dumais, S.T., Hawking, D., Järvelin, K. (eds.) SIGIR, pp. 292–299. ACM, New York (2006) 978  
979  
980
45. Wu, M., Marian, A.: Corroborating answers from multiple web sources. WebDB (2007) 981
46. Yin, X., Han, J., Yu, P.S.: Truth discovery with multiple conflicting information providers on the web. IEEE Trans. Knowl. Data Eng. **20**(6), 796–808 (2008) 982  
983

**AUTHOR QUERIES**

- AQ1. Kindly check the corresponding author.
- AQ2. Please confirm the insertion of ‘with’ in the sentence “...thus associating a semantic label...”
- AQ3. In the sentence “Each each rule extracts...” please check if ‘Each each’ can be changed as ‘Each such’
- AQ4. ‘consensus has been treated as singular. Please check if it is okay in this context.
- AQ5. ‘dependencies’ has been treated as plural. Please check if it is okay in this context.
- AQ6. Please check the sentence “namely, we report source results...”
- AQ7. Please confirm the insertion of Wu et al. in the sentence “A similar direction”
- AQ8. Please check if ‘source dependencies detection’ can be changed as ‘source-dependent detection’

# Chapter 14

## Searching and Browsing Linked Data with SWSE\*

Andreas Harth, Aidan Hogan, Jürgen Umbrich, Sheila Kinsella, Axel Polleres,  
and Stefan Decker

### 14.1 Introduction

Web search engines such as Google, Yahoo! MSN/Bing, and Ask are far from the consummate Web search solution: they do not typically produce direct answers to queries but instead typically recommend a selection of related documents from the Web. We note that in more recent years, search engines have begun to provide direct answers to prose queries matching certain common templates—for example, “population of china” or “12 euro in dollars”—but again, such functionality is limited to a small subset of popular user queries. Furthermore, search engines now provide individual and focused search interfaces over images, videos, locations, news articles, books, research papers, blogs, and real-time social media—although these tools are inarguably powerful, they are limited to their respective domains.

In the general case, search engines are not suitable for complex information gathering tasks requiring aggregation from multiple indexed documents: for such tasks, users must manually aggregate tidbits of pertinent information from various

---

\* Searching and Browsing Linked Data with SWSE saying “The present chapter is an abridged version of [76].

A. Harth (✉)

Karlsruhe Institute of Technology, Institute AIFB, 76128 Karlsruhe, Germany  
e-mail: [harth@kit.edu](mailto:harth@kit.edu)

A. Hogan · J. Umbrich · S. Kinsella · S. Decker

Digital Enterprise Research Institute, National University of Ireland, Galway  
e-mail: [aidan.hogan@deri.org](mailto:aidan.hogan@deri.org); [juergen.umbrich@deri.org](mailto:juergen.umbrich@deri.org); [sheila.kinsella@deri.org](mailto:sheila.kinsella@deri.org);  
[stefan.decker@deri.org](mailto:stefan.decker@deri.org)

A. Polleres

Digital Enterprise Research Institute, National University of Ireland, Galway  
Siemens AG Österreich, Siemensstrasse 90, 1210 Vienna, Austria  
e-mail: [axel@polleres.net](mailto:axel@polleres.net)

pages. In effect, such limitations are predicated on the lack of machine-interpretable structure in HTML documents, which is often limited to generic markup tags mainly concerned with document rendering and linking. Most of the real content is contained in prose text which is inherently difficult for machines to interpret. Addressing the problem of automated interpretation of HTML documents, the semantic Web movement provides a stack of technologies for publishing machine-readable data on the Web, the core of the stack being the resource description framework (RDF).

Using URIs to name things—and not just documents—RDF offers a standardized and flexible framework for publishing structured data on the Web such that (1) data can be linked, incorporated, extended, and reused by other RDF data across the Web; (2) heterogeneous data from independent sources can be automatically integrated by software agents; and (3) the meaning of data can be well defined using ontologies described in RDF using the RDF Schema (RDFS) and Web ontology language (OWL) standards.

Thanks largely to the “Linking Open Data” project [14]—which has emphasized more pragmatic aspects of semantic Web publishing—a rich lode of open RDF data now resides on the Web: this “Web of Data” includes content exported from, for example, Wikipedia, the BBC, the New York Times, Flickr, Last.fm, scientific publishing indexes, biomedical information, and governmental agencies. Assuming large-scale adoption of high-quality RDF publishing on the Web, the question is whether a search engine indexing RDF feasibly could improve upon current HTML-centric engines. Theoretically at least, such a search engine could offer advanced querying and browsing of structured data with search results automatically aggregated from multiple documents and rendered directly in a clean and consistent user interface, thus reducing the manual effort required of its users. Indeed, there has been much research devoted to this topic, with various incarnations of (mostly academic) RDF-centric Web search engines emerging—e.g., Swoogle, Falcons, Watson, Sindice—and in this chapter, we present the culmination of over 6 years of research on another such engine: the “semantic Web search engine” (SWSE).<sup>1</sup>

Indeed, the realization of SWSE has implied two major research challenges:

1. The system must *scale* to large amounts of data.
2. The system must be *robust* in the face of heterogeneous, noisy, impudent, and possibly conflicting data collected from a large number of sources.

Semantic Web standards and methodologies are not naturally applicable in such an environment; in presenting the design and implementation of SWSE, we show how standard semantic Web approaches can be tailored to meet these two challenging requirements.

As such, we present the core of a system which we demonstrate to provide scale, and which is distributed over a cluster of commodity hardware. Throughout, we focus on the unique challenges of applying standard semantic Web techniques and

---

<sup>1</sup><http://swse.deri.org/>

methodologies and show why the consideration of the source of data is an integral part of creating a system which must be tolerant to Web data—in particular, we show how Linked Data principles can be exploited for such purposes. Also, there are many research questions still very much open with respect to the direction of the overall system, as well as improvements to be made in the individual components; we discuss these as they arise, rendering a road map of past, present, and possible future research in the area of Web search over RDF data.

More specifically, we

- Present high-level related work in RDF search engines (Sect. 14.3)
- Present core preliminaries required throughout the rest of the chapter (Sect. 14.4)
- Present the architecture and modus operandi of our system for offering search and browsing over RDF Web data (Sect. 14.5)
- Detail the high-level design of the off-line index building components, including *crawling* (Sect. 14.6), *consolidation* (Sect. 14.7), *ranking* (Sect. 14.8), *reasoning* (Sect. 14.9), and *indexing* (Sect. 14.10)
- Detail the high-level design of the runtime components, including our (lightweight) query processor (Sect. 14.11) and user interface (Sect. 14.12)
- Conclude with discussion of future directions (Sect. 14.13)

## 14.2 Motivating Example

79

To put later discussion into context, we now give a brief overview of the lightweight functionality of the SWSE system; please note that although our methods and algorithms are tailored for the specific needs of SWSE, many aspects of their implementation, design, and evaluation apply to more general scenarios.

Unlike prevalent document-centric Web search engines, SWSE operates over structured data and holds an entity-centric perspective on search: in contrast to returning links to documents containing specified keywords [19], SWSE returns data representations of real-world entities. While current search engines return search results in different domain-specific categories (Web, images, videos, shopping, etc.), data on the semantic Web is flexibly typed and does not need to follow predefined categories. Returned objects can represent people, companies, locations, proteins—anything people care to publish data about.

In a manner familiar to traditional Web search engines, SWSE allows users to specify keyword queries in an input box and responds with a ranked list of result snippets; however, the results refer to entities not documents. A user can then click on an entity snippet to derive a detailed description thereof. The descriptions of entities are automatically aggregated from arbitrarily many sources, and users can cross-check the source of particular statements presented; descriptions also include inferred data—data that have been derived from the existing data through reasoning. Users can subsequently navigate to associated entities.

The screenshot shows a search results page for the keyword "bill clinton". At the top, there is a search bar with the query "bill clinton" and an "Ok" button. Below the search bar are links for "RSS" and "Bill Clinton". The main content area displays three entities:

- Bill Clinton**  
http://dbpedia.org/resource/Bill\_Clinton  
AmericanHumanitarians CausalAgent100007347 DisbarredAmericanLawyers
- Presidency of Bill Clinton**  
http://dbpedia.org/resource/Presidency\_of\_Bill\_Clinton  
Administrator109770949 CorporateExecutive109966255 Executive110069645  
The United States Presidency of Bill Clinton, also known as the Clinton Administration, was the executive branch of the federal government of the United States from January 20, 1993 to January 20, 2001.
- Chelsea Clinton**  
http://data.nytimes.com/N59388218642507311603  
AlumniOfUniversityCollege,Oxford Alumnus109786338 AmericanVegetarians  
Chelsea Victoria Clinton (born February 27, 1980) is the daughter and only child of Bill Clinton, the 42nd U.S. President, and U.S. Secretary of State Hillary Rodham Clinton.

**Fig. 14.1** Results view for keyword query bill clinton

Figure 14.1 shows a screenshot containing a list of entities returned as 100 a result to the keyword search “bill clinton”—such results pages are 101 familiar to HTML-centric engines, with the addition of result types (e.g., 102 DisbarredAmericanLawyers, AmericanVegetarians). Results are 103 aggregated from multiple sources. Figure 14.2 shows a screenshot of the focus 104 (detailed) view of the Bill Clinton entity, with data aggregated from 54 doc- 105 uments spanning six domains (bbc.co.uk, dbpedia.org, freebase.com, 106 nytimes.com, rdfuze.com, and soton.ac.uk), as well as novel data found 107 through reasoning. 108

### 14.3 State of the Art

109

In this section, we give an overview of the state of the art, first detailing distributed 110 architectures for Web search (Sect. 14.3.1), then discussing related systems in the 111 field of “hidden Web” and “deep Web” (Sect. 14.3.2), and finally describing current 112 systems that offer search and browsing over RDF Web data (Sect. 14.3.3)—for a 113 further survey of the latter, cf. [127]. Please note that we will give further detailed 114 related work in the context of each component throughout the chapter. 115

**Bill Clinton**  
[http://dbpedia.org/resource/Bill\\_Clinton](http://dbpedia.org/resource/Bill_Clinton)

<b>label</b>	<ul style="list-style-type: none"> <li>o Bill Clinton</li> <li>o Clinton, Bill@en</li> </ul>
<b>preferred label</b>	<ul style="list-style-type: none"> <li>o Clinton, Bill@en</li> </ul>
<b>definition</b>	<ul style="list-style-type: none"> <li>o Updated: August 5, 2009 OVERVIEW William Jefferson Clinton was president of the United States from 1993 to 2001. The first president born after World War II, the first Democrat since Franklin D. Roosevelt to be elected to a second term, the first to lead in a post-cold-war world, Mr. Clinton also became the first elected president to be impeached, over his deceptions about an affair with a White House intern half his age. Mr. Clinton was acquitted by the Senate, after a trial that was the most hair-raising of many roller-coaster moments in a volatile career. The episode has undoubtedly shaped his legacy. But Mr. Clinton is also likely to be recalled as the president who presided over -- some will say helped spark -- one of the greatest surges in prosperity and innovation in American history, and who helped make the transition from the standoff of the cold war to an era in which American influence was unchallenged. @en</li> </ul>
<b>name</b>	
<b>is in scheme</b>	<ul style="list-style-type: none"> <li>o <a href="#">nytd_per</a></li> </ul>
<b>sameAs</b>	<ul style="list-style-type: none"> <li>o <a href="#">N3779652101308368773</a></li> <li>o <a href="#">clinton_bill_per</a></li> <li>o <a href="#">42nd President of the United States</a></li> <li>more...</li> </ul>
<b>type</b>	<ul style="list-style-type: none"> <li>o <a href="#">SpatialThing</a></li> <li>o <a href="#">Person</a></li> <li>o <a href="#">Concept</a></li> <li>more...</li> </ul>
<b>subject</b>	<ul style="list-style-type: none"> <li>o <a href="#">1946 births</a></li> <li>o <a href="#">20th-century presidents of the United States</a></li> <li>o <a href="#">American Rhodes scholars</a></li> <li>more...</li> </ul>
<b>homepage</b>	<ul style="list-style-type: none"> <li>o <a href="#">Bill+Clinton</a></li> </ul>
<b>depiction</b>	

Fig. 14.2 Focus view for entity Bill Clinton

### 14.3.1 Distributed Web Search Architectures

116

Distributed architectures have long been common in traditional information retrieval-based Web search engines, incorporating distributed crawling, ranking, indexing, and query-processing components [18, 19]. More recent publications relate to the MapReduce framework [31] and to the underlying BigTable [24] distributed database system.

Similar system architectures have been defined in the literature, including WebBase [68] which includes an incremental crawler, storage manager, indexer, and query processor; in particular, the authors focus on hash- and log-based partitioning for storing incrementally updated vast repositories of Web documents. The authors of [97] also describe a system for building a distributed inverted index (based on an embedded database system) over a large corpus of Web pages, for subsequent analysis and query processing.

Much of the work presented herein is loosely inspired by such approaches and thus constitutes an adaptation of such works for the purposes of search over structured data. Since we consider replication, fault tolerance, incremental indexing, etc., currently out of scope, many of our techniques are more lightweight than those discussed.

121

122

123

124

125

126

127

128

129

130

131

132

133

### 14.3.2 Hidden Web/Deep Web Approaches

134

So-called “hidden Web” or “deep Web” approaches [22] are predicated on the premise that a vast amount of the information available on the Web is veiled behind sites with heavy dynamic content, usually backed by relational databases. Such information is largely impervious to traditional crawling techniques since content is usually generated by means of bespoke flexible queries; thus, traditional search engines can only skim the surface of such information [64]. In fact, such data-rich sources have lead to early speculative work on entity-centric search [29].

Approaches to exploit such sources heavily rely on manually constructed, site-specific wrappers to extract structured data from HTML pages [22] or to communicate directly with the underlying database of such sites [25]. Some works have also looked into automatically crawling such hidden-Web sources, by interacting with forms found during traditional crawls [114]; however, this approach is “task specific” and not appropriate for general crawling.

The semantic Web may represent a future direction for bringing deep Web information to the surface, leveraging RDF as a common and flexible data model for exporting the content of such databases, leveraging RDFS and OWL as a means of describing the respective schemata, and thus allowing for automatic integration of such data by Web search engines. Efforts such as D2R(Q) [13] seem a natural fit for enabling RDF exports of such online databases.

### 14.3.3 RDF-Centric Search Engines

154

Early prototypes using the concepts of ontologies and semantics on the Web include Ontobroker [32] and SHOE [65], which can be seen as predecessors to standardization efforts such as RDFS and OWL, describing how data on the Web can be given in structured form and subsequently crawled, stored inferred, and queried over.

Swoogle<sup>2</sup> offers search over RDF documents by means of an inverted keyword index and a relational database [38]. Swoogle calculates metrics that allow ontology designers to check the popularity of certain properties and classes. In contrast to SWSE, which is mainly concerned with entity search over instance data, Swoogle is mainly concerned with more traditional document search over ontologies.

Watson<sup>3</sup> also provides keyword search facilities over semantic Web documents but additionally provides search over entities [30,115]. However, they do not include components for consolidation or reasoning and instead focus on providing APIs to external services.

<sup>2</sup><http://swoogle.umbc.edu/>

<sup>3</sup><http://watson.kmi.open.ac.uk/WatsonWUI/>

Sindice<sup>4</sup> is a registry and lookup service for RDF files based on Lucene and a MapReduce framework [104]. Sindice originally focused on providing an API for finding documents which reference a given RDF entity or given keywords—again, document-centric search. More recently, however, Sindice has begun to offer entity search in the form of Sig.ma<sup>5</sup> [120]. However, Sig.ma maintains a one-to-one relationship between keyword search and results, representing a very different user-interaction model to that presented herein.

The Falcons search engine<sup>6</sup> offers entity-centric searching for entities (and concepts) over RDF data [28]. They map certain keyword phrases to query relations between entities and also use class hierarchies to quickly restrict initial results. Conceptually, this search engine most closely resembles our approach. However, there are significant differences in how the individual components of SWSE and Falcons are designed and implemented. For example, like us, they also rank entities, but using a logarithm of the count of documents in which they are mentioned—we employ a link-based analysis of sources. Also, Falcons supports reasoning involving class hierarchies, whereas we apply a more general rule-based approach, applying a scalable subset of OWL 2 RL/RDF rules. Such differences will be discussed further throughout the chapter.

Aside from domain-agnostic search systems, we note that other systems focus on exploiting RDF for the purposes of domain-specific querying; for example, the recent GoWeb system<sup>7</sup> demonstrates the benefit of searching structured data for the biomedical domain [36]. However, in catering for a specific domain, such systems do not target the same challenges and use cases as we do.

## 14.4 Preliminaries

193

Before we continue, we briefly introduce some standard notation used throughout the chapter—relating to RDF terms (constants), triples, and quadruples—and also discuss Linked Data principles. Note that we will generally use boldface to refer to infinite sets: e.g.,  $\mathbf{G}$  refers to the set of all triples; we will use calligraphy font to denote a subset thereof: e.g.,  $\mathcal{G}$  is a particular set of triples, where  $\mathcal{G} \subset \mathbf{G}$ .

### 14.4.1 Resource Description Framework

200

The resource description framework provides a structured means of publishing information describing entities through use of RDF terms and RDF triples and

<sup>4</sup><http://sindice.com/>

<sup>5</sup><http://sig.ma/>

<sup>6</sup><http://iws.seu.edu.cn/services/falcons/>

<sup>7</sup><http://gopubmed.org/goweb/>

constitutes the core data model for our search engine. In particular, RDF allows for 203  
 optionally defining names for entities using URIs and allows for subsequent reuse 204  
 of URIs across the Web; using triples, RDF allows to group entities into named 205  
 classes, allows to define named relations between entities, and allows for defining 206  
 named attributes of entities using string (literal) values. We now briefly give some 207  
 necessary notation. 208

*RDF Constant.* Given a set of URI references  $\mathbf{U}$ , a set of blank nodes  $\mathbf{B}$ , and a set of 209  
 literals  $\mathbf{L}$ , the set of *RDF constants* is denoted by  $\mathbf{C} = \mathbf{U} \cup \mathbf{B} \cup \mathbf{L}$ . The set of blank 210  
 nodes  $\mathbf{B}$  is a set of existentially quantified variables. The set of literals is given as 211  
 $\mathbf{L} = \mathbf{L}_p \cup \mathbf{L}_d$ , where  $\mathbf{L}_p$  is the set of *plain literals* and  $\mathbf{L}_d$  is the set of *typed literals*. 212  
 A typed literal is the pair  $l = (s, d)$ , where  $s$  is the lexical form of the literal and  $d \in \mathcal{U}$  213  
 is a datatype URI. The sets  $\mathbf{U}$ ,  $\mathbf{B}$ ,  $\mathbf{L}_p$ , and  $\mathbf{L}_d$  are pairwise disjoint. 214

Please note that we treat blank nodes as their skolem versions, that is, not as 215  
 existential variables, but as denoting their own syntactic form. We also ensure 216  
 correct merging of RDF graphs [63] by using blank node labels unique for a given 217  
 source. 218

For URIs, we use namespace prefixes as common in the literature—the full URIs 219  
 can be retrieved from the convenient <http://prefix.cc/> service. For space reasons, we 220  
 sometimes denote `owl :` as the default namespace. 221

*RDF Triple.* A triple  $t = (s, p, o) \in (\mathbf{U} \cup \mathbf{B}) \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{B} \cup \mathbf{L})$  is called an 222  
*RDF triple*. In a triple  $(s, p, o)$ ,  $s$  is called subject,  $p$  predicate, and  $o$  object. 223

*RDF Graph.* We call a finite set of triples an *RDF graph*  $\mathcal{G} \subset \mathbf{G}$  where  $\mathbf{G} =$  224  
 $(\mathbf{U} \cup \mathbf{B}) \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{B} \cup \mathbf{L})$ . 225

*RDF Entity.* We refer to the referent of a URI or blank-node as an *RDF entity*, or 226  
 commonly just entity. 227

#### 14.4.2 Linked Data

228

To cope with the unique challenges of handling diverse and unverified Web data, 229  
 many of our components and algorithms require inclusion of a notion of provenance: 230  
 consideration of the source of RDF data found on the Web. Tightly related to such 231  
 notions are the Linked Data best practices (here paraphrasing [9]): 232

- LDP1** use URIs to name things 233
- LDP2** use HTTP URIs so that those names can be looked up 234
- LDP3** provide useful structured information when a lookup on a URI is made— 235  
 loosely, called *dereferencing* 236
- LDP4** include links using external URIs 237

In particular, within SWSE, these best practices form the backbone of various 238  
 algorithms designed to interact with and be tolerant to Web data. 239

We must thus extend RDF triples with *context* to denote the source thereof [52, 240  
56]. We also define some relations between the identifier for a data source and the 241  
graph it contains, including a function to represent HTTP redirects prevalently used 242  
in Linked Data for **LDP3** [9]. 243

*Data Source.* We define the *http-download* function  $\text{get} : \mathbf{U} \rightarrow 2^{\mathbf{G}}$  as the mapping 244  
from a URI to an RDF graph it may provide by means of a given HTTP lookup [46] 245  
which directly returns status code 200 OK and data in a suitable RDF format.<sup>8</sup> 246  
We define the set of *data sources*  $\mathbf{S} \subset \mathbf{U}$  as the set of URIs  $\mathbf{S} = \{s \in \mathbf{U} \mid \text{get}(s) \neq 247$   
 $\emptyset\}$ . We define the *reference function*  $\text{refs} : \mathbf{C} \rightarrow 2^{\mathbf{S}}$  as the mapping from an RDF 248  
term to the set of data sources that mention it. 249

*RDF Triple in Context/RDF Quadruple.* A pair  $(t, c)$  with a triple  $t = (s, p, o)$ ,  $c \in \mathbf{S}$  250  
and  $t \in \text{get}(c)$  is called a *triple in context*  $c$ . We may also refer to  $(s, p, o, c)$  as an 251  
*RDF quadruple* or quad  $q$  with context  $c$ . 252

*HTTP Dereferencing.* We define *dereferencing* as the function  $\text{deref} : \mathbf{U} \rightarrow \mathbf{U}$  253  
which maps a given URI to the identifier of the document returned by HTTP lookup 254  
operations upon that URI following redirects (for a given finite and noncyclical 255  
path) [46] or which maps a URI to itself in the case of failure. The function 256  
involves stripping the fragment identifier of a URI [11]. Note that we do not dis- 257  
tinguish between the different 30x redirection schemes. All HTTP level functions 258  
 $\{\text{get}, \text{refs}, \text{deref}\}$  are set at the time of the crawl and are bounded by the knowledge 259  
of our crawl: for example,  $\text{refs}$  will only consider documents accessed by the crawl. 260

## 14.5 System Architecture

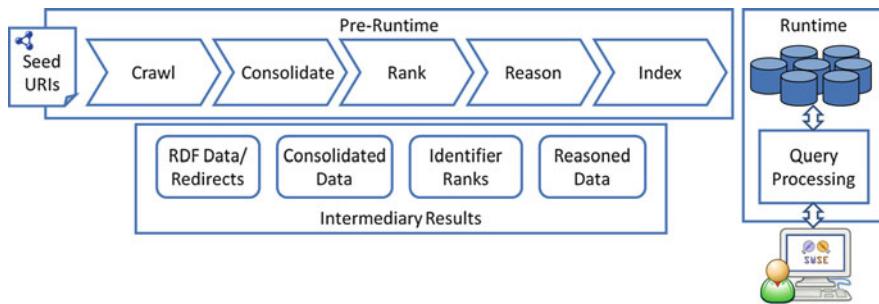
261

The high-level system architecture of SWSE loosely follows that of traditional 262  
HTML search engines [18, 19]. Figure 14.3 illustrates the pre-runtime architecture 263  
of our system, showing the components involved in achieving a local index 264  
of RDF Web data amenable for search. Similar to traditional search engines, 265  
SWSE contains components for crawling, ranking, and indexing data; however, 266  
there are also components specifically designed for handling RDF data, namely, 267  
the consolidation component and the reasoning component. The high-level index 268  
building process is as follows: 269

- The crawler accepts a set of seed URIs and retrieves a large set of RDF data from 270  
the Web. 271
- The consolidation component tries to find synonymous (i.e., equivalent) identi- 272  
fiers in the data and canonicalizes the data according to the equivalences found. 273
- The ranking component performs link-based analysis over the crawled data and 274  
derives scores indicating the importance of individual elements in the data (the 275

---

<sup>8</sup> $2^{\mathbf{G}}$  refers to the power set of  $\mathbf{G}$ .



**Fig. 14.3** System architecture

ranking component also considers URI redirections encountered by the crawler 276  
when performing the link-based analysis). 277

- The reasoning component materializes new data which are implied by the 278 inherent semantics of the input data (the reasoning component also requires URI 279 redirection information to evaluate the trustworthiness of sources of data). 280
- The indexing component prepares an index which supports the information 281 retrieval tasks required by the user interface. 282

Subsequently, the query-processing and user interface components service queries 283 over the index built in the previous steps. 284

Our methods follow the standards relating to RDF [95], RDFS [63], and 285 OWL [116] and leverage the Linked Data principles [9] which state how RDF 286 should be published on the Web. As such, our methods should be sound with 287 respect to data correctly published according to these documents, but we note that 288 oftentimes, the noise inherent in heterogenous RDF Web data may create unintended 289 results. We characterize these problems as they occur, although we know of no 290 method for accurately determining the amount of incorrect or unwanted results 291 generated by these tasks—we note that such considerations may also be subjective 292 (e.g., see [53]). 293

In order to scale, we deploy each of our components over a distributed frame- 294 work, which is based on a shared nothing architecture [117] and consists of one 295 master machine which orchestrates the given tasks and several slave machines which 296 perform parts of the task in parallel. We note that our distribution framework resem- 297 bles the MapReduce framework [31]. Distribution of our algorithms is facilitated 298 by the fact that the preprocessing algorithms are based on scan and sort operations. 299 We omit detailed description of the distributed versions of our algorithms due to 300 space constraints; see [76] for an in-depth treatment of distribution, including full- 301 scale evaluation of each component. 302

## 14.6 Crawling

303

We now begin the discussion of the first component required for building the index, 304  
and thus, for retrieving the raw RDF documents from the Web, that is, the *crawler*. 305  
Our crawler starts with a set of seed URIs, retrieves the content of URIs, parses 306  
and writes content to disk in the form of quads, and recursively extracts new URIs 307  
for crawling. We leverage Linked Data principles (see Sect. 14.4.2) to discover new 308  
sources, where following LDP2 and LDP3, we consider all `http:` protocol URIs 309  
extracted from an RDF document as candidates for crawling. 310

We identify the following requirements for crawling:

- *Politeness*: The crawler must implement politeness restrictions to avoid hammering remote servers with dense HTTP GET requests and to abide by policies identified in the provided `robots.txt` files.<sup>9</sup> 312  
313  
314
- *Throughput*: The crawler should crawl as many URIs as possible in as little time 315  
as is possible within the bounds of the politeness policies. 316
- *Scale*: The crawler should employ scalable techniques and on-disk indexing as 317  
required. 318
- *Quality*: The crawler should prioritize crawling URIs it considers to be “high 319  
quality.” 320

Thus, the design of our crawler is inspired by related work from traditional 321  
HTML crawlers. Additionally—and specific to crawling structured data—we iden- 322  
tify the following requirement: 323

- *Structured Data*: The crawler should retrieve a high percentage of RDF/XML 324  
documents and avoid wasted lookups on unwanted formats, for example, HTML 325  
documents. 326

Currently, we crawl for RDF/XML syntax documents—RDF/XML is still the 327  
most commonly used syntax for publishing RDF on the Web, and we plan in future 328  
to extend the crawler to support other formats such as RDFa, N-Triples, and Turtle. 329

### 14.6.1 High-Level Approach

330

Our high-level approach is to perform breath-first crawling, following precedent set 331  
by traditional Web crawlers (cf. [15, 67]): the crawl is conducted in rounds, with 332  
each round crawling a *frontier*. On a high level, Algorithm 21 represents this round- 333  
based approach applying ROUNDS number of rounds. The frontier comprises of 334  
seed URIs for round 0 (Algorithm 21, line 21), and thereafter with novel URIs 335  
extracted from documents crawled in the previous round (Algorithm 21, line 21). 336  
Thus, the crawl emulates a breadth-first traversal of interlinked Web documents. 337

---

<sup>9</sup><http://www.robotstxt.org/orig.html>

**Algorithm 21:** Algorithm for crawling

---

```

input: SEEDS, ROUNDS, MIN-DELAY
frontier ← SEEDS ;
pld0..n ← new queue;
stats ← new stats;
while rounds + 1 < ROUNDS do
    put frontier into pld0..n;
    for i = 0 to n do
         $\downarrow$  prioritise(pldi, stats);
        start ← current_time();
        for i = 0 to n do
            curi = calculate_cur(pldi, stats);
            if curi > random([0,1]) then
                get uri from pldi;
                urideref = deref(uri);
                if urideref = uri then
                    G = get(uri);
                    output G;
                    UG ← URIs in G;
                    UG ← prune blacklisted from UG ;
                    add unseen URIs in  $\overline{U_G}$  to frontier ;
                    update stats wrt.  $\overline{U_G}$ ;
                else
                    if urideref is unseen then
                        add urideref to frontier;
                        update stats for urideref;
             $\downarrow$ 
            elapsed ← current_time() - start ;
            if elapsed < MIN-DELAY then
                 $\downarrow$  wait(MIN-DELAY - elapsed) ;

```

---

As the bottleneck for a single-threaded crawler will be the response times of remote servers, our implementation of the crawling algorithm is multithreaded and performs concurrent HTTP lookups. Note that the algorithm is further tailored according to requirements we will describe as the section progresses. 338  
339  
340  
341

**14.6.1.1 Incorporating Politeness**

342

The crawler must be careful not to bite the hands that feed it by hammering the servers of data providers or breaching policies outlined in the provided robots.txt file [118]. We use pay-level domains [91] (PLDs; a.k.a. “root domains”; e.g., [bbc.co.uk](http://bbc.co.uk)) to identify individual data providers, and implement politeness on a per-PLD basis. First, when we first encounter a URI for a PLD, we cross-check the robots.txt file to ensure that we are permitted to crawl that site; second, we implement a “minimum PLD delay” to avoid hammering servers, viz., a minimum time period between subsequent requests to a given PLD (MIN-DELAY in Algorithm 21). 343  
344  
345  
346  
347  
348  
349  
350  
351

In order to accommodate the min-delay policy with minimal effect on performance, we must refine our crawling algorithm: large sites with a large internal branching factor (large numbers of unique intra-PLD outlinks per document) can result in the frontier of each round being dominated by URIs from a small selection of PLDs. Thus, naïve breadth-first crawling can lead to crawlers hammering such sites; conversely, given a politeness policy, a crawler may spend a lot of time idle waiting for the min-delay to pass.

One solution is to reasonably restrict the branching factor [91]—the maximum number of URIs crawled per PLD per round—which ensures that individual PLDs with large internal fan-out are not hammered; thus, in each round of the crawl, we implement a cutoff for URIs per PLD, given by PLD-LIMIT in Algorithm 21.

Second, to ensure the maximum gap between crawling successive URIs for the same PLD, we implement a per-PLD queue (given by  $pld_{0..n}$  in Algorithm 21), whereby each PLD is given a dedicated queue of URIs filled from the frontier, and during the crawl, a URI is polled from each PLD queue in a round-robin fashion. If all of the PLD queues have been polled before the min-delay is satisfied, then the crawler must wait: this is given by lines 21–21 in Algorithm 21. Thus, the minimum crawl time for a round—assuming a sufficiently full queue—becomes MIN-DELAY \* PLD-LIMIT.

#### 14.6.1.2 On-Disk Queue

371

As the crawl continues, the in-memory capacity of the machine will eventually be exceeded by the capacity required for storing URIs [91]. Performing a stress test, we observed that with 2 GB of Java heap-space, the crawler could crawl approximately 199 k URIs (additionally storing the respective frontier URIs) before throwing an out-of-memory exception. To scale beyond the implied main-memory limitations of the crawler, we implement on-disk storage for URIs, with the additional benefit of maintaining a persistent state for the crawl and thus offering a “continuation point” useful for extension of an existing crawl, or recovery from failure.

We implement the on-disk storage of URIs using Berkeley DB which comprises of two indexes—the first provides lookups for URI strings against their status (polled/unpolled); the second offers a key-sorted map which can iterate over unpolled URIs in decreasing order of inlink count. The inlink count reflects the total number of documents from which the URI has been extracted thus far; we deem a higher count to roughly equate to a higher priority URI.

The on-disk index and in-memory queue are synchronized at the start of each round:

1. Links and respective inlink counts extracted from the previous round (or seed URIs if the first round) are added to the on-disk index.
2. URIs polled from the previous round have their status updated on-disk.
3. An in-memory PLD queue is filled using an iterator of on-disk URIs sorted by descending inlink count.

The above process ensures that only the URIs active (current PLD queue 393 and frontier URIs) for the current round must be stored in memory. Finally, the 394 in-memory PLD queue is filled with URIs sorted in order of inlink count, offering a 395 cheap form of intra-PLD URI prioritization (Algorithm 21, line 21). 396

#### 14.6.1.3 Crawling RDF/XML

397

Since our architecture is currently implemented to index RDF/XML, we would 398 feasibly like to maximize the ratio of HTTP lookups which result in RDF/XML 399 content; that is, given the total HTTP lookups as  $L$  and the total number of 400 downloaded RDF/XML pages as  $R$ , we would like to maximize the *useful ratio*: 401  $ur = R/L$ . 402

To reduce the amount of HTTP lookups wasted on non-RDF/XML content, we 403 implement the following heuristics: 404

1. First, we blacklist non-`http` protocol URIs. 405
2. Second, we blacklist URIs with common file extensions that are highly unlikely 406 to return RDF/XML (e.g., `html`, `jpg`, `pdf`) following arguments we previously 407 laid out in [121]. 408
3. Third, we check the returned HTTP header and only retrieve the content of URIs 409 reporting `Content-type: application/rdf+xml`.<sup>10</sup> 410
4. Finally, we use a *credible useful ratio* when polling PLDs to indicate the 411 probability that a URI from that PLD will yield RDF/XML based on past 412 observations. 413

With respect to the first two heuristics, any form of URI can be extracted from 414 an RDF/XML document in any position, so from the set of initially extracted 415 URIs, we blacklist those which are non-HTTP or those with common non-RDF 416 file extensions (given in line 21 of Algorithm 21). Although blacklisting URIs with 417 schemes such as `mailto:` or `urn:` is optional (since no HTTP lookup will be 418 performed), pruning them early on avoids the expense of putting them through the 419 queueing process. Note that we do not blacklist HTTP URIs which do not have an 420 explicit file extension and whose content cannot be detected a priori (e.g., those 421 with trailing slashes). Thus, we employ two further heuristics to improve the ratio 422 of RDF/XML. 423

Our third heuristic involves rejecting content based on header information; 424 this is perhaps arguable in that previous observations [74] indicate that 17% 425 of RDF/XML documents are returned with a `Content-type` other than 426 `application/rdf+xml`. Still we automatically exclude such documents from 427 our crawl; however, here we put the onus on publishers to ensure correct reporting 428 of `Content-type`. 429

---

<sup>10</sup>Indeed, one advantage RDF/XML has over RDFa is an unambiguous MIME-type useful in such situations.

With respect to the fourth heuristic, we implement an algorithm for selectively 430  
 polling PLDs based on their observed useful ratio; since our crawler only requires 431  
 RDF/XML, we use this score to access PLDs which offer a higher percentage of 432  
 RDF/XML more often. Thus, we can reduce the amount of time wasted on lookups 433  
 of HTML documents and save the resources of servers for non-RDF/XML data 434  
 providers. 435

The credible useful ratio for PLD  $i$  is derived from the following credibility 436  
 formula: 437

$$cur_i = \frac{rdf_i + \mu}{total_i + \mu}$$

where  $rdf_i$  is the total number of RDF documents returned thus far by PLD  $i$ ,  $total_i$  438  
 is the total number of lookups performed for PLD  $i$  excluding redirects, and  $\mu$  is 439  
 a “credibility factor.” The purpose of the credibility formula is to dampen scores 440  
 derived from few readings (where  $total_i$  is small) toward the value 1 (offering the 441  
 benefit of the doubt), with the justification that the credibility of a score with few 442  
 readings is less than that with a greater number of readings: with a low number of 443  
 readings ( $total_i \ll \mu$ ), the  $cur_i$  score is affected more by  $\mu$  than actual readings for 444  
 PLD  $i$ ; as the number of readings increases ( $total_i \gg \mu$ ), the score is affected more 445  
 by the observed readings than the  $\mu$  factor. Note that we set  $\mu$  to 10.<sup>11</sup> 446

*Example 14.1.* If we observe that PLD  $a = \text{deri.org}$  has returned 1/5 RDF/XML 447  
 documents and PLD  $b = \text{w3.org}$  has returned 1/50 RDF/XML documents, and if 448  
 we assume  $\mu = 10$ , then  $cur_a = (1 + \mu)/(5 + \mu) = 0.7\bar{3}$  and  $cur_b = (1 + \mu)/$  449  
 $(50 + \mu) = 0.18\bar{3}$ . We thus ensure that PLDs are not unreasonably punished for 450  
 returning non-RDF/XML documents early on (i.e., are not immediately assigned a 451  
 $cur$  of 0). 452

To implement selective polling of PLDs according to their useful ratio, we simply 453  
 use the  $cur$  score as a probability of polling a URI from that PLD queue in that 454  
 round (Algorithm 21, lines 21–21). Thus, PLDs which return a high percentage 455  
 of RDF/XML documents—or indeed PLDs for which very few URIs have been 456  
 encountered—will have a higher probability of being polled, guiding the crawler 457  
 away from PLDs which return a high percentage of non-RDF/XML documents. 458

#### 14.6.1.4 PLD Starvation

459

Please note that in our experiments, we have observed the case where there are not 460  
 enough unique PLDs to keep all crawler threads occupied until the MIN-DELAY 461  
 has been reached. We term the state in which crawling threads cannot download 462  
 content in fully parallel fashion *PLD starvation*. Given the politeness restriction 463

---

<sup>11</sup>Admittedly, a “magic number”; however, the presence of such a factor is more important than its actual value: without the credibility factor, if the first document returned by a PLD was non-RDF/XML, then that PLD would be completely ignored for the rest of the crawl.

of, for example, 500 ms per PLD, one PLD with many URIs becomes the hard limit for performance independent of system architecture and crawling hardware, instead imposed by the nature of the Web of Data itself. Also, as a crawl progresses, active PLDs (PLDs with unique content still to crawl) will become less and less, and the performance of the multithreaded crawl will approach that of a single-threaded crawl. As Linked Data publishing expands and diversifies, and as the number of servers offering RDF content increases, better performance would be observed. 464  
465  
466  
467  
468  
469  
470

### 14.6.2 Related Work

471

Parts of our architecture and some of our design decisions are influenced by work on traditional Web crawlers; e.g., the IRLBot system of Lee et al. [91] and the distributed crawler of Boldi et al. [15]. 472  
473  
474

The research field of focused RDF crawling is still quite a young field, with most of the current work based on the lessons learned from the more mature area of traditional Web crawling. Related work in the area of focused crawling can be categorized [7] roughly as follows: 475  
476  
477  
478

- *Classic focused crawling*: e.g., Chakrabarti et al. [23] uses primary link structure and anchor texts to identify pages about a topic using various text similarity of link analysis algorithms. 479  
480  
481
- *Semantic focused crawling*: a variation of classical focused crawling but uses conceptual similarity between terms found in ontologies [40, 41]. 482  
483
- *Learning focused crawling*: Diligenti et al. and Pant and Srinivasan [37, 108] use classification algorithms to guide crawlers to relevant Web paths and pages. 484  
485

However, a major difference between these approaches and ours is that our definition of high-quality pages is not based on topics or ontologies but instead on the content type of documents. 486  
487  
488

With respect to crawling RDF, the Swoogle search engine implements a crawler which extracts links from Google and further crawls based on various—sometimes domain-specific—link extraction techniques [38]; like us, they also use file extensions to throw away non-RDF URIs. In [28], the authors provide a very brief description of the crawler used by the Falcons search engine for obtaining RDF/XML content; interestingly, they provide statistics identifying a power-law type distribution for the number of documents provided by each pay-level domain, correlating with our discussion of PLD starvation. In [115], for the purposes of the Watson engine, the authors use Heritrix<sup>12</sup> to retrieve ontologies using Swoogle, Google, and Protégé indexes and also crawl by interpreting `rdfs:seeAlso` and `owl:imports` as links—they do not exploit the dereferenceability of URIs popularized by Linked Data. Similarly, the Sindice crawler [104] retrieves content 489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500

<sup>12</sup><http://crawler.archive.org/>

based on a *push* model, crawling documents which pinged some central service such 501  
as PingTheSemanticWeb;<sup>13</sup> they also discuss a PLD-level scheduler for ensuring 502  
politeness and diversity of data retrieved. 503

### 14.6.3 Future Directions and Open Research Questions

504

From a pragmatic perspective, we would prioritize extension of our crawler to 505  
handle arbitrary RDF formats—especially the RDFa format which is growing in 506  
popularity. Such an extension may mandate modification of the current mechanisms 507  
for ensuring a high percentage of RDF/XML documents: for example, we could 508  
no longer blacklist URIs with a .html file extension, nor could we rely on the 509  
Content-type returned by the HTTP header (unlike RDF/XML, RDFa does not 510  
have a specific MIME type). 511

Along these lines, we could perhaps also investigate extraction of structured data 512  
from non-RDF sources; these could include microformats, metadata embedded in 513  
documents such as PDFs and images, extraction of HTML metainformation, HTML 514  
scraping, etc. Again, such a process would require revisitaton of our RDF-centric 515  
focused crawling techniques. 516

The other main challenge posed in this section is that of PLD starvation; although 517  
we would expect PLD starvation to become less of an issue as the semantic Web 518  
matures, it perhaps bears further investigation. For example, we have yet to fully 519  
evaluate the trade-off between small rounds with frequent updates of URIs from 520  
fresh PLDs and large rounds which persist with a high delay rate but require less 521  
coordination. Also, given the inevitability of idle time during the crawl, it may 522  
be practical from a performance perspective to give the crawler more tasks to do 523  
in order to maximize the amount of processing done on the data and minimize 524  
idle time. 525

Finally, we have not discussed the possibility of incremental crawls: choosing 526  
URIs to recrawl may lead to interesting research avenues. Besides obvious solutions 527  
such as HTTP caching, URIs could be recrawled based on, e.g., detected change 528  
frequency of the document over time, some quality metric for the document, or how 529  
many times data from that document were requested in the UI. More practically, 530  
an incremental crawler could use PLD statistics derived from previous crawls and 531  
the HTTP headers for URIs—including redirections—to achieve a much higher 532  
ratio of lookups to RDF documents returned. Such considerations would largely 533  
countermand the effects of PLD starvation, by reducing the amount of lookups the 534  
crawler needs in each run. 535

---

<sup>13</sup><http://pingthesemanticweb.com/>

## 14.7 Entity Consolidation

536

In theory, RDF enables excellent data integration over data sourced from arbitrarily many sources—as is the case for our corpora collected by our crawler. However, the integration is premised on the widespread sharing and reuse—across all sources—of URIs for specific entities. In reality, different RDF documents on the Web published by independent parties often speak about the same entities using different URIs [73];<sup>14</sup> to make matters worse, RDF allows for the definition of anonymous entities—entities identified by a blank node—without a prescribed URI.

As an example, in our 1.118 bn statement Linked Data corpus, we found 23 different URIs identifying the person Tim Berners-Lee—these identifiers spanned nine different PLDs.<sup>15</sup> Now, given a keyword query for “tim bernes lee,” the data using each of the 23 different identifiers would be split over 23 different results, even though they all refer to the same entity.

Offering search and querying over a raw RDF dataset collected from the Web would thus entail many duplicate results referring to the same entity, emulating the current situation on the HTML Web where information about different resources is fragmented across source documents. Given a means of identifying *equivalent entities* in RDF data—entities representing the same real-world individual but identified incongruously—would enable the merging of information contributions on an entity given by heterogeneous sources without the need for consistent URI naming of entities.

In fact, OWL [116] provides some standard solutions to such problems. First, OWL defines the `owl:sameAs` property which is intended to relate two equivalent entities; the property has symmetric, transitive, and reflexive semantics as one would expect. Many sources on the Web offer `owl:sameAs` links between entities described locally and equivalent entities described remotely.

Further, OWL provides some other mechanisms for discovering implicit `owl:-sameAs` relations in the absence of explicit relations: the most prominent such example is provision of the class `owl:InverseFunctionalProperty`, which defines a class of properties whose value uniquely identifies an entity. One example of an inverse-functional property would be an ISBN property, where ISBN values uniquely identify books. If two entities share the same ISBN value, a same-as relation can be inferred between them. Using OWL, same-as relations can also be detected using `owl:FunctionalProperty`, `owl:maxCardinality`, and `owl:cardinality` (and now in OWL 2 RL using `owl:maxQualifiedCardinality` and `owl:qualifiedCardinality`); however, the recall of inferences involving the latter OWL constructs are relatively small [75] and thus considered out of scope here.

---

<sup>14</sup>In fact, Linked Data principles could be seen as encouraging this practice, where dereferenceable URIs must be made local.

<sup>15</sup>These equivalent identifiers were found through explicit `owl:sameAs` relations.

In [73], we provided a straightforward batch-processing technique for deriving owl:sameAs relations using inverse-functional properties defined in the data. However, the precision of such inferences is questionable. As an example, in [73] we found 85,803 equivalent individuals to be inferable from a Web dataset through the “void” values 08445a31a78661b5c746feff39a9db6e4e2cc5cf and da39a3ee5e6b4b0d3255bfef95601890afd80709 for the prominent inverse-functional property foaf:mbox\_sha1sum: the former value is the sha1-sum of an empty string and the latter is the sha1-sum of the “mailto:” string, both of which are erroneously published by online Friend-of-a-Friend (FOAF—a very popular vocabulary used for personal descriptions) exporters.<sup>16</sup> Aside from obvious pathological cases—which can of course be blacklisted—publishers commonly do not respect the semantics of inverse-functional properties [74].

More recently, in [70], we showed that we could find  $1.31 \times$  more sets of equivalent identifiers by including reasoning over inverse-functional properties and functional properties, than when only considering explicit owl:sameAs. These sets contained  $2.58 \times$  more identifiers. However, we found that the additional equivalences found through such an approach were mainly between blank nodes on domains which do not use URIs to identify resources, common for older FOAF exporters: we found a 6% increase in URIs involved in an equivalence. We again observed that the equivalences given by such an approach tend to offer more noise than when only considering explicit owl:sameAs relations.

In fact, the performance of satisfactory, high-precision, high-recall entity consolidation over large-scale Linked Data corpora is still an open research question. At the moment, we rely on owl:sameAs relations which are directly asserted in the data to perform consolidation.

### 14.7.1 High-Level Approach

599

The overall approach involves two scans of the main body of data, with the following high-level steps:

- owl:sameAs statements are extracted from the data: the main body of data is scanned once identifying owl:sameAs triples and buffering them to a separate location.
- The transitive/symmetric closure of the owl:sameAs statements are computed, inferring new owl:sameAs relations.
- For each set of equivalent entities found (each *equivalence class*), a *canonical identifier* is chosen to represent the set in the consolidated output.

---

<sup>16</sup>See, for example, <http://blog.livedoor.jp/nkgw/foaf.rdf>

4. The main body of data is again scanned and consolidated: identifiers are rewritten 609  
to their canonical form—we do not rewrite identifiers in the predicate position, 610  
objects of `rdf:type` triples, or literal objects. 611

In previous work, we have presented two approaches for performing such consolidation; in [73], we stored `owl:sameAs` in memory, computing the transitive/ 612  
symmetric closure in memory, and performing in-memory lookups for canonical 613  
identifiers in the second scan. In [75], we presented a batch-processing technique 614  
which uses on-disk sorts and scans to execute the `owl:sameAs` transitive/sym- 615  
metric closure, and the canonicalization of identifiers in the main body of data. 616  
The former approach is in fact much faster in that it reduces the amount of 617  
time consumed by hard-disk I/O operations; however, the latter batch-processing 618  
approach is not limited by the main-memory capacity of the system. Either 619  
approach is applicable with our consolidation component (even in the distributed 620  
case); however, since for the moment we only operate on asserted `owl:sameAs` 621  
statements (we found  $\sim 12\text{ m}$  `owl:sameAs` statements in our full-scale crawl, 622  
which we tested to be within our 4GB in-memory capacity using the flyweight 623  
pattern), for now we apply the faster in-memory approach. 624

Standard OWL semantics mandates duplication of data for all equivalent terms 625  
by the semantics of replacement (cf. Table 4, [50]), which, however, is not a practical 626  
option at scale. First, the amount of duplication will be quadratic with respect to the 627  
size of an equivalence class. Empirically, we have found equivalence classes with 628  
8.5 k elements from explicit `owl:sameAs` relations [76]. If one were to apply 629  
transitive, reflexive, and symmetric closure of equivalence over these identifiers, we 630  
would produce  $8.5k^2 = 72.25m$  `owl:sameAs` statements alone; further assuming 631  
an average of six unique quads for each identifier—51 k unique quads in total— 632  
we would produce a further 433.5 m repetitive statements by substituting each 633  
equivalent identifier into each quad. Second, such duplication of data would result 634  
in multitudinous duplicate results being presented to end users, with obvious impact 635  
on the usability of the system. 636

Thus, the practical solution is to abandon standard OWL semantics and instead 637  
*consolidate* the data by choosing a canonical identifier to represent the output data 638  
for each equivalence class. Canonical identifiers are chosen with preference to URIs 639  
over blank nodes, and thereafter we arbitrarily use a lexicographical order—the 640  
canonical identifiers are only used internally to represent the given entity.<sup>17</sup> Along 641  
these lines, we also preserve all URIs used to identify the entity by outputting 642  
`owl:sameAs` relations to and from the canonical identifier (please note that we 643  
do not preserve redundant blank-node identifiers which are only intended to have a 644  
local scope, have been assigned arbitrary labels during the crawling process, and are 645  
not subject to reuse), which can subsequently be used to display all URIs originally 646  
used to identify an entity, or to act as a “redirect” from an original identifier to the 647  
canonical identifier containing the pertinent information. 648

---

<sup>17</sup>If necessary, the ranking algorithm presented in the next section could be used to choose the most popular identifier as the canonical identifier.

In the in-memory map structure, each equivalence class is assigned a canonical identifier according to the above ordering. We then perform a second scan of the data, rewriting terms according to canonical identifiers. Please note that according to OWL full semantics, terms in the predicate position and object position of `rdf:type` triples should be rewritten (referring to term positions occupied by properties and classes respectively in membership assertions; again, cf. Table 4, [50]). However, we do not wish to rewrite these terms: in OWL, equivalence between properties can instead be specified by means of the `owl:equivalentProperty` construct, and between classes as the `owl:equivalentClass` construct.<sup>18</sup> We omit rewriting class/property terms in membership assertions, handling inferences involving classes/properties by alternate means in Sect. 14.9.

Thus, in the second scan, the subject and object of non-`rdf:type` statements are rewritten according to the canonical identifiers stored in the in-memory map, with rewritten statements written to output. If no equivalent identifiers are found, the statement is buffered to the output. When the scan is complete, `owl:sameAs` relations to/from canonical URIs and their equivalent URIs are appended to the output. Consolidation is now complete.

### 14.7.2 Related Work

668

Entity consolidation has an older related stream of research relating largely to databases, with work under the names of record linkage, instance fusion, and duplicate identification; cf. [26, 98, 103] and a survey at [42]. Due to the lack of formal specification for determining equivalences, these older approaches are mostly concerned with probabilistic methods.

With respect to RDF, Bouquet et al. [17] motivate the problem of (re)using common identifiers as one of the pillars of the semantic Web and provide a framework for mapping heterogeneous identifiers to a centralized naming scheme for reuse across the Web—some would argue that such a centralized service would not be in tune with the architecture or philosophy of the Web.

Consolidation has also been tackled in the context of applying rule-based reasoning for OWL, where choosing canonical (or “pivot”) identifiers is a common optimization [73, 75, 85, 88, 122]. Algorithms for building the equivalence partition have also been proposed by these works, where, e.g., Kolovski et al. propose a similar union–find algorithm to that presented herein. In previous works [75], we have presented an algorithm which iteratively connects elements in an equivalence class toward the common, lowest-ordered element, pruning connections to elements to

<sup>18</sup>As an example of naïve usage of `owl:sameAs` between classes and properties on the Web, please see: [http://colab.cim3.net/file/work/SICoP/DRMITIT/DRM\\_OWL/Categorization/TaxonomyReferenceModel.owl](http://colab.cim3.net/file/work/SICoP/DRMITIT/DRM_OWL/Categorization/TaxonomyReferenceModel.owl)

higher orders; the result of this process is a canonical identifier which connects 686 directly to all elements of the equivalence class. Urbani et al. [122] propose an 687 algorithm which is similar in principle, but which is implemented in a distributed 688 MapReduce setting and optimized to prevent load-balancing issues. We discuss 689 related rule-based reasoning approaches in more detail later in Sect. 14.9.2. 690

The Sindice and Sig.ma search systems internally use inverse-functional prop- 691 erties to find equivalent identifiers [104, 120]—Sindice uses reasoning to identify 692 a wider range of inverse-functional properties [104]. Online systems RKBEx- 693 plorer [49],<sup>19</sup> <sameAs>,<sup>20</sup> and ObjectCoref<sup>21</sup> offer on-demand querying for 694 owl : sameAs relations found for a given input URI, which they internally compute 695 and store; as previously alluded to, the former publish owl : sameAs relations for 696 authors and papers in the area of scientific publishing. 697

Some more recent works have looked at hybrid approaches for consolidation, 698 combining symbolic (i.e., reasoning) methods and statistical/inductive methods 699 [78, 80]. 700

The authors of [124] present *Silk*: a framework for creating and maintaining 701 interlinkage between domain-specific RDF datasets; in particular, this framework 702 provides publishers with a means of discovering and creating owl : sameAs 703 links between data sources using domain-specific rules and parameters. Thereafter, 704 publishers can integrate discovered links into their exports, enabling better linkage 705 of the data and subsequent consolidation by data consumers: this framework goes 706 hand in hand with our approach, producing the owl : sameAs relations which we 707 consume. 708

In [53], the authors discuss the semantics and current usage of owl : sameAs in 709 Linked Data, discussing issues relating to *identity* and providing four categories of 710 owl : sameAs usage to relate entities which are closely related, but for which the 711 semantics of owl : sameAs—particularly substitution—do not quite hold. 712

### 14.7.3 Future Directions and Open Research Questions

713

In this section, we have focused on the performance of what we require to be a 714 scalable consolidation component. We have not presented analysis of the precision 715 or recall of such consolidation—such evaluation is difficult to achieve in practice 716 given a lack of a gold standard, or suitable means of accurately verifying results. 717 The analysis of the precision and recall of various scalable consolidation methods 718 on current Web data would represent a significant boon to research in the area of 719 querying over Linked Data. 720

We are currently investigating statistical consolidation methods, with partic- 721 ular emphasis on extracting some notion of the quality or trustworthiness of 722

<sup>19</sup><http://www.rkbexplorer.com/sameAs/>

<sup>20</sup><http://sameas.org/>

<sup>21</sup><http://ws.nju.edu.cn/objectcoref/>

derived equivalences [78]. Presently, we try to identify “quasi-inverse-functional” 723 and “quasi-functional” properties (properties which are useful for distinguishing 724 identity) using statistical analysis of the input data. We then combine shared 725 property/value pairs for entities and derive a fuzzy value representing the confidence 726 of equivalence between said entities. However, this preliminary work needs further 727 investigation—including scalability and performance testing, and integration with 728 more traditional reasoning-centric approaches for consolidation—before being 729 included in the SWSE pipeline. 730

A further avenue for research in the same vein is applying “disambiguation,” or 731 attempting to assert that two entities cannot (or are likely not to) be equivalent using 732 statistical approaches or analysis of inconsistencies in reasoning: disambiguation 733 would allow for increasing the precision of the consolidation component by quickly 734 removing “obvious” false positives. 735

Again, such approaches would likely have a significant impact on the quality 736 of data integration possible in an engine such as SWSE operating over RDF 737 Web data. 738

## 14.8 Ranking

739

Ranking is an important mechanism in the search process with the function of prioritizing data elements. Herein, we want to quantify the importance of consolidated 740 entities in the data for use in ordering the presentation of results returned when 741 users pose a keyword query (e.g., see Fig. 14.1), such that the most “important” 742 results appear higher in the list. Note that we will combine these ranking scores 743 with relevance scores later in Sect. 14.10. 744

There is a significant body of related work on link-based algorithms for the Web; 745 seminal works include [86, 107]. A principal objective when ranking on the Web 746 is rating popular pages higher than unpopular ones—further, ranks can be used for 747 performing top- $k$  processing, allowing the search engine to retrieve and process 748 small segments of results ordered by their respective rank. Since we share similar 749 goals, we wish to leverage the benefits of link-based analysis, proven for the HTML 750 Web, for the purposes of ranking Linked Data entities. We identify the following 751 requirements for ranking Linked Data, which closely align with those of HTML- 752 centric ranking schemes: 753

- The methods should be *scalable* and applicable in scenarios involving large 755 corpora of RDF. 756
- The methods should be *automatic* and *domain agnostic*, and not inherently 757 favoring a given domain or source of data. 758
- The methods should be *robust* in the face of spamming. 759

With respect to ranking the entities in our corpus in a manner sympathetic with 760 our requirements, we further note the following: 761

- On the level of triples (data level), publishers can provide arbitrary information 762 in arbitrary locations using arbitrary identifiers; thus, to discourage low-effort 763 spamming, the source of information must be taken into account. 764
- Following traditional link-based ranking intuition, we should consider links from 765 one source of information to another as a “positive vote” from the former to the 766 latter. 767
- In the absence of sufficient source-level ranking, we should infer links between 768 sources based on usage of identifiers on the data level and some function mapping 769 between data-level terms and sources. 770
- Data providers who reuse identifiers from other sources should not be penalized: 771 their data sources should not lose any rank value. 772

In particular, our methods are inspired by Google’s PageRank [107] algorithm, 773 which interprets hyperlinks to other pages as positive votes. However, PageRank 774 is generally targeted toward hypertext documents, and adaptation to Linked Data 775 sources is nontrivial, given that the notion of a hyperlink (interpreted as a vote for a 776 particular page) is missing: Linked Data principles mandate *implicit* links to other 777 Web sites or data sources through reuse of dereferenceable URIs. Also, the unit of 778 search is no longer a document, but an entity. 779

In previous work [59], we proposed a scalable algorithm for ranking structured 780 data from an open, distributed environment, based on a concept we term *naming* 781 *authority*. We reintroduce selected important discussion from [59] and extend here 782 by implementing the method in a distributed way and reevaluating with respect to 783 performance. 784

#### 14.8.1 High-Level Approach

785

Although we wish to rank entities, our ranking algorithm must consider the source 786 of information to avoid low-effort data-level spamming. Thus, we must first have a 787 means of ranking source-level identifiers and thereafter can propagate such ranks to 788 the data level. 789

To leverage existing link-based analysis techniques, we need to build a graph 790 encoding the interlinkage of Linked Data sources. Although one could examine use 791 of, for example, owl:imports or rdfs:seeAlso links, and interpret them 792 directly as akin to a hyperlink, the former is used solely in the realm of OWL 793 ontology descriptions and the latter is not restricted to refer to RDF documents; 794 similarly, both ignore the data-level linkage that exists by means of **LDP4** (include 795 links using external URIs). Thus, we aim to infer source-level links through usage 796 of data-level URIs in the corpus. 797

To generalize the idea, we previously defined the notion of “naming authority” 798 for identifiers: a naming authority is a data source with the power to define identifiers 799 of a certain structure [59]. Naming authority is an abstract term which could be 800 applied to a knowable provenance of a piece of information, be that a document, 801

host, person, organization, or other entity. Data items which are denoted by unique  
identifiers may be reused by sources other than the naming authority.

802

803

*Example 14.2.* With respect to Linked Data principles (see Sect. 14.4.2), consider for example the data-level URI <http://danbri.org/foaf.rdf#danbri>. Clearly the owner(s) of the <http://www.danbri.org/foaf.rdf> document (or, on a coarser level, the danbri.org domain) can claim some notion of “authority” for this URI: following **LDP4**, the usage of the URI on other sites can be seen as a vote for the respective data source. We must also support redirects as commonly used for **LDP3**—thus we can reuse the `deref` function given in Sect. 14.4.2 as a function which maps an arbitrary URI identifier to the URI of its naming authority document (or to itself in the absence of a redirect).

804

805

806

807

808

809

810

811

812

Please note that there is no obvious function for mapping from literals to naming authority, we thus omit them from our source-level ranking (one could consider a mapping based on datatype URIs, but we currently see no utility in such an approach). Also, blank nodes may only appear in one source document and are not subject to reuse: although one could reduce the naming authority of a blank node to the source they appear in, clearly only self-links can be created.

813

814

815

816

817

818

Continuing, we must consider the granularity of naming authority: in [59], we discussed and contrasted interpretation of naming authorities on a document level (e.g., <http://www.danbri.org/foaf.rdf>) and on a PLD level (`danbri.org`). Given that the PLD-level linkage graph is significantly smaller than the document-level graph, the overhead for aggregating and analyzing the PLD-level graph is significantly reduced, and thus, we herein perform ranking at a PLD level.

819

820

821

822

823

824

Please note that for convenience, we will assume that PLDs are identified by URIs, for example, (<http://danbri.org/>). We also define the convenient function `pld : U → U` which extracts the PLD identifier for a URI (if the PLD cannot be parsed for a URI, we simply ignore the link)—we may also conveniently use the function `plds : 2U → 2U` for sets of URIs.

825

826

827

828

829

Thus, our ranking procedure consists of the following steps:

830

1. Construct the *PLD-level naming authority graph*: for each URI  $u$  appearing in a triple  $t$  in the input data, create links from the PLDs of sources mentioning a particular URI to the PLD of that URI:  $\text{plds}(\text{refs}(u)) \rightarrow \text{pld}(\text{deref}(u))$ .
2. From the naming authority graph, use the PageRank algorithm to derive scores for each PLD.
3. Using the PLD ranks, derive a rank value for terms in the data, particularly terms in  $\mathbf{U} \cup \mathbf{B}$  which identify entities.

831

832

833

834

835

836

837

## 14.8.1.1 Extracting Source Links

838

As a first step, we derive the naming authority graph from the input dataset. That is, 839  
 we construct a graph which encodes links between data source PLDs, based on the 840  
 implicit connections created via identifier reuse. 841

Given PLD identifiers  $p_i, p_j \in \mathbf{U}$ , we specify the naming authority matrix  $A$  as 842  
 a square matrix defined as: 843

$$a_{i,j} = \begin{cases} 1 & \text{if } p_i \neq p_j \text{ and } p_i \text{ uses an identifier} \\ & \quad \text{with naming authority } p_j \\ 0 & \text{otherwise} \end{cases}$$

This represents an  $n \times n$  square matrix where  $n$  is the number of PLDs in the data, 844  
 and where the element at  $(i, j)$  is set to 1 if  $i \neq j$  and PLD  $i$  mentions a URI which 845  
 leads to a document hosted by PLD  $j$ . 846

As such, the naming authority matrix can be arbitrarily derived through a single 847  
 scan over the entire dataset. Note that we optionally can omit URIs found in the 848  
 predicate position of a triple, or the object position of an `rdf:type` triple, in 849  
 the derivation of the naming authority graph, such that we do not want to overly 850  
 inflate scores for PLDs hosting vocabularies: we are concerned that such PLDs 851  
 (e.g., `w3.org`, `xmlns.org`) would receive rankings orders of magnitude higher 852  
 than their peers, overly inflating the ranks of arbitrary terms appearing in that PLD; 853  
 further, users will generally not be interested in results describing the domain of 854  
 knowledge itself [59]. 855

## 14.8.1.2 Calculating Source Ranks

856

Having constructed the naming authority matrix, we now can compute scores for 857  
 data sources. For computing ranking scores, we perform a standard PageRank 858  
 calculation over the naming authority graph: we calculate the dominant eigenvector 859  
 of the naming authority graph using the Power iteration while taking into account a 860  
 damping factor (see [107] for more details). 861

## 14.8.1.3 Calculating Identifier Ranks

862

Based on the rank values for the data sources, we now calculate the ranks for 863  
 individual identifiers. The rank value of a constant  $c \in \mathbf{C}$  is given as the summation 864  
 of the rank values of the PLDs for the data sources in which the term occurs: 865

$$\text{idrank}(c) = \sum_{\textit{pld} \in \textit{plds}(\textit{refs}(c))} \text{sourcerank}(\textit{pld})$$

This follows the simple intuition that the more highly ranked PLDs mentioning a given term, the higher the rank of that term should be.<sup>22</sup> Note again—and with similar justification as for deriving the named authority graph—we do not include URIs found in the predicate position of a triple, or the object position of an `rdf:type` triple in the above summation for our evaluation. Also note that the ranking for literals may not make much sense depending on the scenario—in any case, we currently do not require ranks for literals.

#### 14.8.1.4 User Evaluation

873

Herein, we summarize the details of our user evaluation, where the full details are available in [59]. We conducted a study asking 10–15 participants to rate the ordering of SWSE results given for five different input keyword queries, including the evaluator’s own name. We found that our method produced preferable results (with statistical significance) for ranking entities than the baseline method of implementing PageRank on the RDF node-link graph (an approach which is similar to existing work such as ObjectRank [6]). Also, we found that use of the PLD-level graph and document-level graph as input for our PageRank calculations yielded roughly equivalent results for identifier ranks in our user evaluation.

882

#### 14.8.2 Related Work

883

There have been numerous works dedicated to comparing hypertext-centric ranking for varying granularity of sources. Najork et al. [101] compared results of the HITS [86] ranking approach when performed on the level of document, host, and domain granularity and found that domain granularity returned the best results: in some cases, PLD-level granularity may be preferable to domain or host-level granularity because some sites like LiveJournal (which export vast amounts of user profile data in the Friend-of-a-Friend [FOAF] vocabulary) assign subdomains to each user, which would result in large tightly-knit communities if domains were used as naming authorities. Previous work has performed PageRank on levels other than the page level, for example, at the more coarse granularity of directories, hosts, and domains [83], and at a finer granularity such as logical blocks of text [21] within a page.

895

There have been several methods proposed to handle the task of ranking semantic Web data.

897

Swoogle ranks documents using the OntoRank method, a variation on PageRank which iteratively calculates ranks for documents based on references to terms

899

---

<sup>22</sup>The generic algorithm can naturally be used to propagate PLD/source-level rankings to any form of RDF artifact, including triples, predicates, classes, etc.

(classes and properties) defined in other documents. We generalize the method 900  
described in [39] to rank entities and perform link analysis on the PLD abstraction 901  
layer. 902

ObjectRank [6] ranks a directed labeled graph using PageRank using “authority 903  
transfer schema graphs,” which requires manual weightings for the transfer of 904  
propagation through different types of links; further, the algorithm does not include 905  
consideration of the source of data and is perhaps better suited to domain-specific 906  
ranking over verified knowledge. 907

We note that Falcons [28] also rank the importance of entities (what they call 908  
“objects”), but based on a logarithm of the number of documents in which the object 909  
is mentioned. 910

In previous work, we introduced ReConRank [72]: an initial effort to apply 911  
a PageRank-type algorithm to a graph which unifies data-level and source-level 912  
linkage. ReConRank does take data provenance into account; however, because it 913  
simultaneously operates on the object graph, it is more susceptible to spamming 914  
than the presented approach. 915

A recent approach for ranking Linked Data called Dataset rankING (DING) 916  
[35]—used by Sindice—holds a similar philosophy to ours: they adopt a two- 917  
layer approach consisting of an entity layer and a dataset layer. However, they also 918  
apply rankings of entities within a given dataset, using PageRank (or optionally 919  
link-counting) and unsupervised link-weighting schemes, subsequently combining 920  
dataset and local entity ranks to derive global entity ranks. Because of the local 921  
entity ranking, their approach is theoretically more expensive and less flexible than 922  
ours, but would offer better granularity of results—less entity results with the same 923  
rank. However, as we will see later, we will be combining global entity ranks with 924  
keyword query-specific relevance scores, which mitigates the granularity problem. 925

There are numerous other loosely related approaches, which we briefly mention: 926  
SemRank [3] ranks relations and paths on semantic Web data using information- 927  
theoretic measures; AKTiveRank [1] ranks ontologies based on how well they cover 928  
specified search terms; Ontocopi [2] uses a spreading activation algorithm to locate 929  
instances in a knowledge base which are most closely related to a target instance; 930  
the SemSearch system [92] also includes relevance ranks for entities according to 931  
how well they match a user query. 932

### 14.8.3 Future Directions and Open Research Questions

933

Ranking in Web search engines depends on a multitude of factors, ranging from 934  
globally computed ranks to query-dependent ranks to location, preferences, and 935  
history of the searcher. Factoring additional signals into the ranking procedure 936  
is the area for further research, especially in the face of complex database-like 937  
queries and results beyond the simple list of objects. For example, we have already 938  
seen that we exclude predicate and class identifiers from the ranking procedure in 939  
order not to adversely affect our goal of ranking entities (individuals) in the data; 940

specific modes and display criteria of the UI may require different models of ranks, 941  
providing multiple contextual ranks for identifiers in different roles—e.g., creating 942  
a distinctive ranking metric for identifiers in the role of predicates, reflecting the 943  
expectations of users given various modes of browsing. 944

Another possibly fruitful research topic relates to the question of finding 945  
appropriate mathematical representations of directed labeled graphs and appropriate 946  
operations on them [47, 58]. Most of the current research in ranking RDF graphs 947  
is based around the directed graph models borrowed from hypertext ranking 948  
procedures. A bespoke mathematical model for RDF (directed, labeled, and named) 949  
graphs may lead to a different view on possible ranking algorithms. 950

Finally, the evaluation of link-based ranking as an indicator of trustworthiness 951  
would also be an interesting contribution; thus far, we have evaluated the approach 952  
according to user evaluation reflecting preference for the prioritization of entity 953  
results in the UI. However, given that we also consider the source of information 954  
in our ranking, we could see if there was a co-occurrence, for example, of poorly 955  
ranked PLDs and inconsistent data. Such a result would have impact for the 956  
reasoning component, presented next, and some discussion is provided in the 957  
respective future work section to follow. 958

## 14.9 Reasoning

959

Using the Web ontology language (OWL) and the RDF Schema language (RDFS), 960  
instance data (i.e., assertional data) describing individuals can be supplemented with 961  
structural data (i.e., terminological data) describing classes and properties, allowing 962  
to well define the domain of discourse and ultimately provide machines a more 963  
sapient understanding of the RDF data. Numerous vocabularies have been published 964  
on the Web of Data, encouraging reuse of terms for prescribed classes and properties 965  
across sources and providing formal RDFS/OWL descriptions thereof. 966

We have already seen that OWL semantics can be used to automatically 967  
aggregate heterogeneous data—using `owl:sameAs` relations and, for example, the 968  
`owl:InverseFunctionalProperty` to derive `said`—where the knowledge 969  
is fractured by the use of discordant identifiers.<sup>23</sup> However, RDFS and OWL 970  
descriptions in the data can be further exploited to infer new statements based on the 971  
terminological knowledge and provide a more complete dataset for query answering 972  
and to automatically translate data from one conceptual model to another (where 973  
appropriate mappings exist in the data). 974

*Example 14.3.* In our data, we find 43 properties whose memberships can be 975  
used to infer an `foaf:page` relationship between a resource and a Web page 976  
pertaining to it. These include specializations of the property within the FOAF 977

<sup>23</sup>Note that in this chapter, we deliberately decouple consolidation and reasoning, since in future work we hope to view the unique challenges of finding equivalent identifiers as separate from those of inferencing according to terminological data presented here.

namespace itself, such as `foaf:homepage`, `foaf:weblog`, etc., and specializations of the property outside the FOAF namespace, including `mo:wikipedia`, `rail:arrivals`, `po:microsite`, `plink:rss`, `xfn:mePage`, etc. All such specializations of the property are related to `foaf:page` (possibly indirectly) through the `rdfs:subPropertyOf` relation in their respective vocabulary. Similarly, *inverses* of `foaf:page` may also exist, where in our corpus we find that `foaf:topic` relates a Web page to a resource it pertains to. Here, `foaf:topic` is related to `foaf:page` using the built-in OWL property `owl:inverseOf`. Thus, if we know that:

```
ex:resource mo:wikipedia ex:wikipage .  
mo:wikipedia rdfs:subPropertyOf foaf:page .  
foaf:page owl:inverseOf foaf:topic .
```

we can *infer* through *reasoning* that:

```
ex:resource foaf:page ex:wikipage .  
ex:wikipage foaf:topic ex:resource .
```

In particular, through the RDFS and OWL definitions given in the data, we infer a new fact about the entities `ex:resource` and `ex:wikipage`. (Note that reasoning can also apply over class memberships in a similar manner.)

We identify the following requirements for large-scale RDFS and OWL reasoning over Web data:

- *Precomputation*: the system should precompute inferences to avoid the runtime expense of backward chaining such that could negatively impact upon response times.
- *Reduced output*: the system should not produce so many inferences that it overburdens the consumer application.
- *Scalability*: the system should scale near linearly with respect to the size of the Linked Data corpus.
- *Web tolerant*: the system should be tolerant to noisy and possibly inconsistent data on the Web.
- *Domain agnostic*: the system should be applicable over data from arbitrary domains and consider noncore Web ontologies (ontologies other than RDF(S)/OWL) as equals.

In previous work [75], we introduced the scalable authoritative OWL reasoner (SAOR) system for performing large-scale materialization using a rule-based approach over a fragment of OWL, according to the given requirements. We subsequently generalized our approach, extended our fragment to a subset of OWL 2 RL/RDF, and demonstrated distributed execution in [77]. We now briefly reintroduce important aspects from that work, focusing on discussion relevant to the SWSE use case.

### 14.9.1 High-Level Approach

1017

First, we choose a rule-based approach which offers greater tolerance in the 1018  
inevitable event of inconsistency than description logics-based approaches—indeed, 1019  
consistency cannot be expected on the Web (cf. [74] for our discussion on reasoning 1020  
issues in Linked Data). Second, rule-based approaches offer greater potential 1021  
for scale following arguments made in [45]. Finally, many Web ontologies— 1022  
although relatively lightweight and inexpressive—are not valid DL ontologies: for 1023  
example, FOAF defines the data-type property `foaf:mbox_sha1sum` as inverse- 1024  
functional, which is disallowed in OWL DL—in [8] and [125], the authors provided 1025  
surveys of Web ontologies and showed that most are in OWL full, albeit for largely 1026  
syntactic reasons. 1027

However, there does not exist a standard ruleset suitable for application over 1028  
arbitrary Web data—we must compromise and deliberately abandon completeness, 1029  
instead striving for a more pragmatic form of reasoning tailored for the 1030  
unique challenges of Web reasoning [69]. In [75], we discussed the tailoring of a 1031  
nonstandard OWL ruleset—viz. pD\* [79]—for application over Web data. More 1032  
recently, OWL 2 has become a W3C Recommendation, and interestingly from our 1033  
perspective includes a standard rule-expressible fragment of OWL, viz.: OWL 2 1034  
RL [50]. In [77], we presented discussion on the new ruleset from the perspective of 1035  
application over Web data, and showed that the ruleset is not immediately amenable 1036  
to the requirements outlined, and still needs amendment for our purposes. We also 1037  
refer the interested reader to [70] for more detail on use cases and techniques for 1038  
applying OWL reasoning over Linked Data. 1039

First, from the OWL 2 RL/RDF ruleset, we do not apply rules which specifically 1040  
infer what we term as “tautological statements,” which refer to syntactic RDFS and 1041  
OWL statements such as `rdf:type rdfs:Resource` statements and reflexive 1042  
`owl:sameAs` statements—statements which apply to every term in the graph. 1043  
Given  $n$  rules which infer such statements and  $t$  unique terms in the dataset, such 1044  
rules would burden the consumer application with  $t * n$  largely jejune statements—in 1045  
fact, we go further and filter such statements from the output. 1046

Second, we identified that separating terminological data (our T-Box)<sup>24</sup> that 1047  
describes classes and properties from assertional data (our A-Box) that describes 1048  
individuals could lead to certain optimizations in rule execution, leveraging the 1049  
observation that only <1% of Linked Data are terminological and that the termino- 1050  
logical data are the most frequently accessed segment for OWL reasoning [75]. 1051  
We used such observations to justify the identification, separation, and provision of 1052  
optimized access to our T-Box, storing it in memory. 1053

Third, after initial evaluation of the system at scale encountered a puzzling 1054  
deluge of inferences, we discovered that incorporating the source of data into the 1055

---

<sup>24</sup>For example, we consider the triples `mo:wikipedia rdfs:subPropertyOf foaf:page`  
and `foaf:page owl:inverseOf foaf:topic` to be terminological.

reasoning algorithm is of utmost importance; naively applying reasoning over the merge of arbitrary RDF graphs can lead to unwanted inferences whereby third parties redefine classes and properties provided by popular ontologies [75]. For example, one document<sup>25</sup> defines owl:Thing to be a member of 55 union classes and another defines nine *properties* as the domain of rdf:type.<sup>26</sup> We counteract such behavior by incorporating the analysis of authoritative sources for classes and properties in the data.

We will now discuss the latter two issues in more detail, but beforehand, let us treat some preliminaries used in this section.

#### 14.9.1.1 Reasoning Preliminaries

We briefly reintroduce some notions formalized in [75, 77]; for brevity, in this section, we aim to give an informative and informal description of terms and refer the interested reader to [75, 77] for a more formal description thereof.

*Generalized Triple.* A *generalized triple* is a triple where blank nodes and literals are allowed in all positions [50]. Herein, we assume generalized triples internally in the reasoning process and postfilter non-RDF statements from the output.

*Metaclass.* Informally, we consider a *metaclass* as a class specifically of classes or properties; i.e., the members of a metaclass are themselves either classes or properties. Herein, we restrict our notion of meta-classes to the set defined in RDF(S) and OWL specifications, where examples include rdf:Property, rdfs:Class, owl:Class, owl:Restriction, owl:DatatypeProperty, owl:-FunctionalProperty, etc.; rdfs:Resource, rdfs:Literal, e.g., are not metaclasses.

*MetaproPERTY.* A *metaproPERTY* is one which has a metaclass as its domain; again, we restrict our notion of metaproPERTIES to the set defined in RDF(S) and OWL specifications, where examples include rdfs:domain, rdfs:-subClassOf, owl:hasKey, owl:inverseOf, owl:oneOf, owl:on-Property, owl:unionOf, etc.; rdf:type, owl:sameAs, rdfs:label, e.g., do not have a metaclass as domain.

*Terminological Triple.* We define the set of *terminological triples* as the union of the following sets of generalized triples:

1. Triples with rdf:type as predicate and a metaclass as object
2. Triples with a metaproPERTY as predicate
3. Triples forming a *valid* RDF list whose head is the object of a metaproPERTY (e.g., a list used for owl:unionOf, owl:intersectionOf)

<sup>25</sup><http://lsdis.cs.uga.edu/~oldham/ontology/wsag/wsag.owl>

<sup>26</sup><http://www.eiao.net/rdf/1.0>

*Example 14.4.* The triples:

```
mo:wikipedia rdfs:subPropertyOf foaf:page .  
foaf:page owl:inverseOf foaf:topic .
```

are considered terminological, whereas the following are not:

```
ex:resource mo:wikipedia ex:wikipage .  
ex:resource rdf:type rdfs:Resource .
```

*Triple Pattern.* A *triple pattern* is a generalized triple where variables from the infinite set  $\mathbf{V}$  are allowed in all positions. We call a set (to be read as conjunction) of triple patterns a *basic graph pattern*. Following standard notation, we prefix variables with “?” We say that a triple is a *binding* of a triple pattern if there exists a mapping of the variables in the triple pattern to some set of RDF constants such that, subsequently, the triple pattern equals the triple; we call this mapping *variable binding*. The notion of a binding for a graph pattern follows naturally.

*Terminological/Assertional Pattern.* We refer to a *terminological-triple/terminological-graph pattern* as one that can only be bound by a terminological triple or, resp., a set thereof. An *assertional pattern* is any pattern which is not terminological.

*Inference Rule.* We define an *inference rule*  $r$  as the pair  $(\text{Ante}, \text{Con})$ , where the antecedent  $\text{Ante}$  and the consequent  $\text{Con}$  are basic graph patterns [112]; all variables in  $\text{Con}$  are contained in  $\text{Ante}$ , and if  $\text{Ante}$  is nonempty, at least one variable must coexist in  $\text{Ante}$  and  $\text{Con}$ . Every unique match—in the union of the input and inferred data—for the graph pattern  $\text{Ante}$  leads to the inference of  $\text{Con}$  with the respective variable bindings. Rules with empty  $\text{Ante}$  can be used to model axiomatic statements which hold for every graph. Herein, we use SPARQL-like syntax to represent graph patterns, and will typically formally write inference rules as  $\text{Ante} \Rightarrow \text{Con}$ .

*Example 14.5.* The OWL 2 RL/RDF rule *prp-spo1* [50] supports inferences for  $\text{rdfs:subPropertyOf}$  with the following rule:

```
?p1 rdfs:subPropertyOf ?p2 . ?x ?p1 ?y . ⇒ ?x ?p2 ?y .
```

where the antecedent  $\text{Ante}$  consists of the two patterns on the left side of  $\Rightarrow$  and the consequent  $\text{Con}$  consists of the pattern on the right side of  $\Rightarrow$ . This can be read as an IF–THEN condition, where if data matching the patterns on the left are found, the respective bindings are used to infer the respective pattern on the right.

#### 14.9.1.2 Separating Terminological Data

Given the above preliminaries, we can now define our notion of a  *$\mathcal{T}$ -split inference rule*, whose antecedent is split into two: one part which can only be matched by terminological data and one which can be matched by assertional data.

**Definition 14.1 ( $\mathcal{T}$ -split inference rule).** Let  $r$  be the rule  $(\text{Ante}, \text{Con})$ . We define the  $\mathcal{T}$ -split version of  $r$  as the triple  $(\text{Ante}_{\mathcal{T}}, \text{Ante}_{\mathcal{G}}, \text{Con})$ , where  $\text{Ante}_{\mathcal{T}}$  is the set of terminological patterns in  $\text{Ante}$  and  $\text{Ante}_{\mathcal{G}}$  is given as all remaining antecedent patterns:  $\text{Ante} \setminus \text{Ante}_{\mathcal{T}}$ .

We generally write  $(\text{Ante}_{\mathcal{T}}, \text{Ante}_{\mathcal{G}}, \text{Con})$  as  $\underline{\text{Ante}_{\mathcal{T}}} \underline{\text{Ante}_{\mathcal{G}}} \Rightarrow \text{Con}$ , identifying terminological patterns by underlining.

*Example 14.6.* Take the rule *prp-dom* [50]:

$$\text{?p rdfs:domain ?c . ?x ?p ?y .} \Rightarrow \text{?y rdf:type ?c .}$$

The terminological (underlined) pattern can only be matched by triples who have  $\text{rdfs:domain}$ —a metaproPERTY—as predicate, and thus must be terminological. The second pattern can be matched by nonterminological triples and so is considered an assertional pattern.

Given the general notion of terminological data, we can constrain our *T-Box* (Terminological-Box) to be the set of terminological triples present in our input data that match a terminological pattern in our rules—intuitively, our T-Box represents the descriptions of classes and properties required in our ruleset; e.g., if our ruleset is RDFS, we do not include OWL terminological triples in our T-Box. We define our *closed T-Box*—denoted  $\mathcal{T}$ —as the set of terminological triples derived from the input and the result of exhaustively applying rules with no assertional patterns (axiomatic and “schema-level” rules) up to a *least fixed point* [77]. Again, our “A-Box” is the set of all statements, including the *T-Box* and inferred statements.

When applying a  $\mathcal{T}$ -split inference rule,  $\text{Ante}_{\mathcal{T}}$  is strictly only matched by our closed T-Box. Thus, in our reasoning system, we have a well-defined distinction between T-Box and A-Box information, reflected in the definition of our rules, and the application of rules over the T-Box split data. This decoupling of T-Box and A-Box allows for incorporating the following optimizations:

1. Knowing that the T-Box is relatively small and is the most frequently accessed segment for reasoning, we can store the T-Box in an optimized index.
2. We can identify optimized  $\mathcal{T}$ -split rules as those with low assertional-arity—namely, rules which do not require joins over a large A-Box can be performed in an optimal and scalable manner.
3. The separation of the T-Box enables straightforward distribution over commodity hardware.

With respect to the first possible optimization, at the moment, we store the entire T-Box in memory. With respect to the second optimization, rules involving more than one assertional pattern (i.e., requiring a join operation over the large A-Box) are in practice difficult to compute at the necessary scale [75]. We thus categorize rules according to the *assertional arity* of their antecedent; i.e., the number of assertional patterns in the antecedent. In [71], we performed similar categorization of OWL 2 RL/RDF rules. We then apply a subset of OWL 2 RL/RDF rules with only one assertional pattern; these rules do not require joins to be performed within

the large A-Box but rather only between the T-Box and the A-Box. With regard 1168  
to the third optimization for distribution, the T-Box can be replicated on each 1169  
machine, minimizing the amount of data exchange and messages sent between 1170  
machines [77]. 1171

Assuming that we apply rules with only one assertional pattern, we can apply the 1172  
following high-level reasoning procedure: 1173

1. To commence, we apply rules with no antecedent, inferring axiomatic statements. 1174
2. We then run the first scan of the data, identifying terminological knowledge found 1175  
in the data and separating and indexing the data in our in-memory T-Box. 1176
3. Using this T-Box, we apply rules which only require T-Box knowledge, deriving 1177  
the closed T-Box. 1178
4. The second scan sequentially joins individual A-Box statements with the static 1179  
in-memory T-Box, including recursively inferred statements. 1180

We call the above reasoning approach “partial indexing” in that only a subset of 1181  
the data needs to be indexed for lookups. In general, the partial-indexing approach is 1182  
suitable when only a small subset of the data need be indexed. In the above version, 1183  
rules without A-Box joins are not supported, so we need only to index the T-Box. 1184  
The approach is sound with respect to standard exhaustive rule application (e.g., 1185  
semiautomatic evaluation) and also complete with the condition that a rule requiring 1186  
assertional knowledge does not infer terminological triples (our T-Box is static and 1187  
will not be updated) [77]. In summary, by avoiding expensive intra-A-Box joins, we 1188  
instead perform reasoning at roughly the cost of two sequential scans of the input 1189  
data and the cost of writing the inferred statements to disk [75]. 1190

AQ1

#### 14.9.1.3 Authoritative Reasoning

1191

In order to curtail the possible side-effects of open Web data publishing, we include 1192  
the source of data in inferencing. Our methods are based on the view that a publisher 1193  
instantiating a vocabulary’s term (class/property) thereby accepts the inferencing 1194  
mandated by that vocabulary and recursively referenced vocabularies for that term. 1195  
Thus, once a publisher instantiates a class or property from a vocabulary, only that 1196  
vocabulary and its references should influence what inferences are possible through 1197  
that instantiation. 1198

In order to do so, we again leverage Linked Data best practices—in this case, 1199  
particularly **LDP2** and **LDP3**—use HTTP URIs, and offer an entity description at 1200  
the dereferenced document. Similar to the ranking procedure, we follow the intuition 1201  
that the document returned by resolving a URI is authoritative for that URI, and the 1202  
prerogative of that document on that URI should have special consideration. More 1203  
specifically—and recalling the dereferencing function `deref` and HTTP lookup 1204  
function `get` from Sect. 14.4—we can define the authoritative function which 1205  
gives the set of terms for which a graph at a given Web location (source) speaks 1206  
authoritatively: 1207

$$\begin{aligned}
 auth : S \rightarrow 2^C \\
 s \mapsto & \{ b \in B \mid b \in t \in \text{get}(u) \} \\
 & \cup \{ u \in U \mid \text{deref}(u) = s \}
 \end{aligned}$$

where a Web document is authoritative for the blank nodes it contains and the URIs which dereference to it; for example, the FOAF vocabulary is authoritative for terms in its namespace. Note that no document is authoritative for literals.

Now we wish to perform reasoning over terms as mandated in the respective authoritative document. For example, we want to perform inferencing over data instantiating FOAF classes and properties as mandated by the FOAF vocabulary and not let third-party vocabularies (not recursively referenced by FOAF) affect said inferencing. To negate the effects of nonauthoritative axioms on reasoning over Web data, we apply restrictions to the  $\mathcal{T}$ -split application of rules with nonempty  $Ante^T$  and  $Ante^G$ , whereby the document serving the T-Box data bound by  $Ante^T$  must be authoritative for at least one term bound by a variable which appears in both  $Ante^T$  and  $Ante^G$ : that is to say, the document serving the terminological data must speak authoritatively for at least one term in the assertional data being reasoned over. We call the nonauthoritative redefinition of third-party terms and remote vocabularies “ontology hijacking” [75].

*Example 14.7.* Take the OWL 2 RL/RDF rule **cax-sco**:

$$\text{?c1 rdfs:subClassOf ?c2 . ?x a ?c1 .} \Rightarrow \text{?x a ?c2 .}$$

where we use `a` as a shortcut for `rdf:type`. Here, `?c1` is the only variable that appears in both  $Ante^T$  and  $Ante^G$ . Take an A-Box triple

$$\text{ex:me a foaf:Person .}$$

Here, `?c1` is bound by `foaf:Person`, and  $\text{deref}(\text{foaf:Person}) = \text{foaf:}$ , the FOAF spec. Now, any document serving a binding for

$$\text{foaf:Person rdfs:subClassOf ?c2 .}$$

must be authoritative for the term `foaf:Person`: the triple must come from the FOAF spec. Note that `?c2` need not be authoritatively bound; e.g., FOAF can *extend* any classes they like.

We do not consider authority for rules with empty  $Ante^T$  or  $Ante^G$ . Also, we consider reasoned T-Box triples as nonauthoritative, thus *effectively* excluding these triples from the T-Box [75].

Since authoritativeness is on a T-Box level, we can effectively prefilter terminological triples for each rule based on the source providing them. We refer the reader to [70, 75] for more detail on authoritative reasoning, including empirical analysis of the explosion of inferences encountered without the notion of authority. Note that the previous two examples from documents in footnotes 25 and 26 are ignored by the authoritative reasoning.

### 14.9.2 Related Work

1243

We have extended our reasoning algorithm toward larger coverage of OWL 2 RL/RDF and parallel distribution of inference [76]. Similarly, other works have been presented that tackle large-scale reasoning through parallelization: Urbani et al. [123] presented a MapReduce approach to RDFS reasoning in a cluster of commodity hardware similar to ourselves, identifying that RDFS rules have, at most, one assertional pattern in the antecedent, discussing how this enables efficient MapReduce support. Published at the same venue, Weaver and Hendler [126] also leverage a separation of terminological data to enable distribution of RDFS reasoning. Although the above works have demonstrated scale in the order of hundreds of millions and a billion triples respectively, their experiments were focused on scalability issues and not on counteracting poor data quality on the Web. Weaver et al. [126] focus on evaluation over synthetic LUBM data; Urbani et al. [123] apply RDFS over  $\sim 865$  m Linked Data triples, but produce 30 bn inferences which is against our requirement of reduced output—they do not consider authoritative reasoning or source of data, although they note in their performance-centric paper that an algorithm similar to that in SAOR could be added.

A number of systems have tackled the distributed computation of A-Box joins. The MARVIN [106] system uses a “divide-conquer-swap” technique for performing joins in a distributed setting, avoiding hash-based data partitioning to avoid problems with data skew inherent in RDF [89]. Following on from [123], Urbani et al. introduced the WebPie system [122], applying incomplete but comprehensive pD\* to 100 bn LUBM triples, discussing rule-specific optimizations for performing pD\* “A-Box join rules” over MapReduce.

In more recent work [88], Kolovski et al. have presented an (Oracle) RDBMS-based OWL 2 RL/RDF materialization approach. They again use some similar optimizations to the scalable reasoning literature, including parallelization, canonicalization of `owl:sameAs` inferences, and also partial evaluation of rules based on highly selective patterns—from discussion in the chapter, these selective patterns seem to correlate with the terminological patterns of the rule. Unlike the approaches mentioned thus far, the authors tackle the issue of updates, proposing variants of seminaive evaluation to avoid rederivations.

Although these works are certainly a large step in the right direction, we feel that applying such rules over 1 bn triples of arbitrary Linked Data is still an open research question given our previous experiences documented in [75]: for example, applying full and quadratic materialization of transitive inferences over the A-Box may become infeasible.(if not now, then almost certainly in the future).

With respect to template rules, DLEJena [96] uses the Pellet DL reasoner for T-Box level reasoning and uses the results to template rules for the Jena rule engine; they only demonstrate methods on synthetic datasets up to a scale of  $\sim 1$  m triples. We take a somewhat different approach, discussing template rules in the context of the partial indexing technique, giving a lightweight bottom-up approach to optimizations.

A viable alternative approach to Web reasoning employed by Sindice [33]—the relation to which is discussed in depth in [75]—is to consider a small “per-document” closure which quarantines reasoning to a given document and the related documents it either implicitly or explicitly imports. Although such an approach misses inferences made through the merge of documents—for example, transitivity across sources—so does ours given our current limitation of not computing A-Box joins. 1286  
1287  
1288  
1289  
1290  
1291  
1292

Falcons employ a similar approach to our authoritative analysis to do reasoning over class hierarchies, but only include support of `rdfs:subClassOf` and `owl:equivalentClass`, as opposed to our general framework for authoritative reasoning over arbitrary  $\mathcal{T}$ -split rules [27]. 1293  
1294  
1295  
1296

### 14.9.3 Future Directions and Open Research Questions

1297

In order to make reasoning over arbitrary Linked Data feasible—both in terms of scale and usefulness of the inferred data—we currently renounce a lot of inferences theoretically warranted by the OWL semantics. We would thus like to extend our approach to cover a more complete fragment of OWL 2 RL/RDF, while still meeting the requirements outlined. This would include, for example, a cost–benefit analysis of rules which require A-Box joins for reasoning over Web data. Similarly, since we perform partial materialization—and indeed since full OWL 2 RL/RDF materialization over Linked Data will probably not be feasible—we would like to investigate some backward-chaining (runtime) approaches which complement a partial materialization strategy. Naturally, such extensions would push the boundaries for scalability and performance even further than our current, cautious approach. 1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309

Finally, we have not considered the meaningful assignment of context to inferred triples. Currently in SWSE, each inference is assigned a placeholder context which denotes the last rule fired in its derivation. Thus, the provenance of inferences is lost. Currently, since our inferences are derived from a single assertional triple, we could consider assigning the inference the same context as that triple. If more than one document is considered responsible for an inference (be it terminological or assertional), tracking the provenance of inferences then becomes much more complicated, possibly leading to a large growth in quadruples required to store all possible provenances. 1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318

Relatedly, we do not currently consider the combination of ranking into the reasoning process, where ranking is currently applied before (and independently of) reasoning. In more exploratory works [16, 70], we have extended our approach to include some notion of ranking, incorporating the ranks of triples and their contexts (using a variation of the algorithm in Sect. 14.8) into inference, and investigating the applicability of ranking as a quantification of the trustworthiness of inferences. We use these ranks to repair detected *inconsistencies*: contradictions present in the corpus. In particular, we found  $\sim 301$  k inconsistencies after reasoning, although 1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326

~294 k of these were given by invalid datatypes, with ~7 k members of disjoint classes. Along similar lines, inclusion of ranking could be used to facilitate top- $k$  materialization: for example, only materializing triples relating to popularly instantiated classes and properties. Integration of these methods into the SWSE pipeline is the subject of future work.

1327  
1328  
1329  
1330  
1331

## 14.10 Indexing

1332

Having now reached the end of the discussion on the data acquisition, analysis, and enhancement components, we look at creating an index necessary to allow users perform top- $k$  keyword lookups and focus lookups (see Sect. 14.2) over the Linked Data crawl which has been consolidated and includes the results of the reasoning process. Note that in previous work, we demonstrated a distributed system for allowing SPARQL querying over billions of triples [60]; however, we deem SPARQL out of scope for this work, focusing instead on a lightweight, bespoke index optimized for the requirements of the user interface.

1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340

To allow for speedy access to the RDF data, we employ a set of indexes: an inverted index for keyword lookups based on RDF literals (text) and a sparse index for lookups of structured data. Inverted indexes are standard for keyword searches in information retrieval. For structured data, we use a sparse index because it represents a good trade-off between lookup performance, scalability, and simplicity [60]. Following our previous techniques aiming at application over static datasets, our index structure does not support updates and is instead read optimized [60]; in principle, we could employ any sufficiently optimized implementation of an index structure that offers prefix lookup capabilities on keys.

1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349

### 14.10.1 Inverted Index

1350

The inverted index is required to formulate the direct response to a user keyword query to be rendered by the UI (again, see Fig. 14.1). Our inverted index for keyword search is based on the Lucene [62]<sup>27</sup> engine and is constructed in the following ways during a scan of the data:

1351  
1352  
1353  
1354

- For each entity in the RDF graph, construct a Lucene document with the union of all string literals related by some property to the RDF subject.
- To each entity, add fields containing the identifiers (URI(s) or blank node given by the subject and/or owl:sameAs values), labels (rdfs:label, dc:title, etc.), descriptions (rdfs:comment, dc:description, etc.),

1355  
1356  
1357  
1358  
1359

<sup>27</sup><http://lucene.apache.org/java/>

classes (objects of `rdf:type` triples), and possibly other metadata such as 1360  
image URIs if required to create keyword result snippets. 1361

- For each identifier, add globally computed ranks. 1362

For lookups, we specify a set of keyword terms for which matching identifiers 1363  
should be returned and, in addition, the desired slice of the result set (e.g., `result` 1364  
`1 to 10`). Following standard information retrieval techniques, Lucene combines 1365  
the globally computed ranks with query-dependent *TF\*IDF* (query-relevance) ranks 1366  
and selects the slice of results to be returned. We additionally associate entity 1367  
labels with a fixed “boost” score, giving label-term matches higher relevance, here 1368  
assuming that many keyword searches will be for entity labels (e.g., `galway`, `dan` 1369  
`brickley`). We use Lucene’s off-the-shelf similarity engine [62] which can be 1370  
sketched as follows. 1371

Let  $q$  be the query,  $t$  a keyword term,  $e_c$  the entity with the canonical identifier 1372  
 $c$ ,  $\text{lit}_p(e_c)$  the set of literals attached to  $e_c$  by the predicate  $p$ ,  $\text{lit}(e_c)$  the set of all 1373  
literals attached to  $e_c$ , and  $r_c = \text{idrank}(c)$  the global (identifier) rank for that entity. 1374  
The score of an entity  $e_c$  with respect to the query  $q$  is then computed as follows: 1375

$$\text{score}(q, e_c) = \sum_{l \in \text{lit}_p(e_c)} \sum_{t \in q \cap l} (tf_{t \in \text{lit}(e_c)} * idf_t^2 * b_p) * r_c$$

where  $b_p$  is a weighting factor for different predicates which we use to boost label 1376  
fields; where 1377

$$tf_{t \in l} = \sqrt{\text{freq}_t(\text{lit}(e_c))}$$

represents the term frequency, given here as the square root of the number of 1378  
appearances of  $t$  in all literals attached to  $e_c$ ; and where

$$idf_t = 1 + \log\left(\frac{n}{n_t + 1}\right)$$

represents the inverse document frequency, where  $n$  is the total number of entities 1378  
indexed and  $n_t$  is the total number of entities associated with the term  $t$ . 1379

The additional nontextual metadata stored in Lucene allows for result snippets 1380  
to be directly created from the Lucene results, without requiring access to the 1381  
structured index: from the contents of the additional fields, we generate an RDF 1382  
graph and return the results to higher layers for generating the results page. 1383

## 14.10.2 Structured Index

1384

The structured index is implemented to give all information relating to a given entity 1385  
(e.g., focus view; again see Fig. 14.2). The structured index is implemented using 1386

“sparse indexes” [60], where a blocked and sorted ISAM file contains the RDF quads and lookups are supported by a small in-memory index which holds the first entry of each block: binary search is performed on the in-memory index to locate the on-disk blocks which can potentially contribute to the answer, where subsequently, those blocks are fetched, parsed, and answers filtered and returned. Currently, we only require lookups on the subject position of quads, and thus only require one index sorted according to the natural order  $(s, p, o, c)$ . 1387  
1388  
1389  
1390  
1391  
1392  
1393

There are two tuning parameters for such an index. The first is block size, which determines (1) the size of the chunks of data fetched from disk and (2) indirectly the size of the in-memory portion of the index. The second parameter is compression: minimizing the amount of data transferred from disk to memory should speed up lookups, provided that the time saved by smaller data transfers outweighs the time required for uncompressing data. 1394  
1395  
1396  
1397  
1398  
1399

### 14.10.3 Related Work

1400

A veritable plethora of RDF stores have been proposed in the literature, most aiming at providing SPARQL functionality, and each bringing with it its own set of priorities for performance, and its own strengths and weaknesses. A subset of these systems rely on underlying relation databases for storage, including 4store [54], Bigdata®,<sup>28</sup> Hexastore [128], Jena SDB,<sup>29</sup> Mulgara,<sup>30</sup> Sesame [20], Virtuoso [43], etc.; the rest rely on so-called native RDF storage schemes, including HPRD [93], Jena TDB,<sup>31</sup> RDF3X [102], SIREn [34], Voldemort,<sup>32</sup> etc. 1401  
1402  
1403  
1404  
1405  
1406  
1407

We note that many SPARQL engines include inverted indexes—usually Lucene-based—to offer keyword search over RDF data. The authors of [99] describe full-text search benchmarking of existing RDF stores—in particular Jena, Sesame2, Virtuoso, and YARS2—testing queries of varying degrees of complexity involving full-text search. They showed that for many types of queries, the performance of YARS2 was often not as competitive as other stores, and correctly verified that certain types of queries (e.g., keyword matches for literals of a given property) are not supported by our system. With respect to performance, we have only ever implemented naïve full-SPARQL query-optimization techniques in YARS2, and have instead focused on creating scalable read-optimized indexes, demonstrating batch processing of joins in a distributed environment and focusing on efficiently supporting simple lookups which potentially return large result sets. For example, we choose not to use OIDs (internal integer representations of constants): although 1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420

<sup>28</sup><http://www.systap.com/bigdata.htm>

<sup>29</sup><http://openjena.org/SDB/>

<sup>30</sup><http://www.mulgara.org/>

<sup>31</sup><http://openjena.org/TDB/>

<sup>32</sup><http://project-voldemort.com/>

OIDs are a proven avenue for optimized query processing involving large amounts 1421  
of intermediate results (e.g., cf. [102]), we wish to avoid the expensive translation 1422  
from internal OIDs to potentially many external constants, instead preserving the 1423  
ability to stream results directly. In general, we do not currently require support 1424  
for complex structured queries, and question the utility of more complex full-text 1425  
functionality to lay users. 1426

#### **14.10.4 Future Directions and Open Research Questions**

The future work of the indexing section is inextricably linked with that of the future 1428  
direction of the query processing and user interface components. At the moment, our 1429  
index supports simple lookups for entities matching a given keyword, data required 1430  
to build a keyword snippet, and the quads for which that subject appears. 1431

Given a relatively static query model, a custom-built structured index can be 1432  
tailored to offer optimized service to the user interface, as opposed to, e.g., a generic 1433  
SPARQL engine. The main directions for future work in indexing would be to 1434  
identify an intersection of queries for which optimized indexes can be built in a 1435  
scalable manner and queries which offer greater potential to the UI. 1436

Further investigation of compression techniques and other low-level optimizations 1437  
may further increase the base performance of our system—however, we 1438  
feel that the combination of RLE encoding and GZIP compression currently 1439  
demonstrates satisfactory performance. 1440

A recent trend in data management is the emergence of so-called NoSQL 1441  
databases which offer distributed indexing. A possible avenue of further research 1442  
involves systems following the BigTable [24] data model such as Apache Cassan- 1443  
dra [90] which could offer distributed indexing capability for RDF. 1444

With regard to the combination of ranking factors in the keyword index, we 1445  
currently use static boosting in the off-the-shelf Lucene similarity engine. Investigat- 1446  
ing different methodologies for combining query-dependent and query-independent 1447  
rankings (and across multiple fields given by different properties) is thus an open 1448  
question. For example, in recent work, Pérez-Agüera et al. have recently claimed 1449  
that BM25F offers better results when combining multiple structured fields for 1450  
indexing keywords in RDF [109]. 1451

### **14.11 Query Processing**

With the distributed index built and prepared on the slave machines, we now require 1453  
a query processor to accept user queries; request and orchestrate lookups over 1454  
the slave machines, and aggregate, process, and stream the final results. In this 1455  
section, we assume that the master machine hosts the query processor: however, 1456

we look at different configurations in Sect. 14.12. We aim to characterize the query-processing steps, and give performance for sequential lookups over the distributed index, and for the various information-retrieval tasks required by the user interface. In particular, we describe the two indexes needed for processing user keyword queries and user focus queries respectively (see Sect. 14.2).

1457  
1458  
1459  
1460  
1461

### 14.11.1 Distributed Keyword-Query Processing

1462

For a top- $k$  keyword query, the coordinating machine requests  $k$  result identifiers and ranks from each of the slave machines. The coordinating machine then computes the aggregated top- $k$  hits and requests the snippet result data for each of the hits from the originating machines and streams data to the initiating agent. For the purposes of pagination, given a query for page  $n$ , the originating machine requests the top  $n * k$  result identifiers and associated ranks and then determines, requests, and streams the relevant result snippets.

1463  
1464  
1465  
1466  
1467  
1468  
1469

### 14.11.2 Distributed Focus-Query Processing

1470

Creating the raw data for the focus view of a given entity is somewhat complicated by the requirements of the UI. The focus view mainly renders the information encoded by quads for which the identifier of the entity appears in the subject position; however, to provide a more legible rendering, the UI requires human-readable labels for each predicate and object, as well as ranks for prioritizing elements in the rendered view (see predicate/object labels in Fig. 14.2). Thus, to provide the raw data required for the focus view of a given entity, the master machine accepts the relevant identifier, performs a hash function on the identifier, and directly requests data from the respective slave machine (which itself performs a lookup on the structured index). Subsequently, the master machine generates a unique set of predicates and objects appearing in the result set; this set is then split by hash, with each subset sent in parallel to the target slave machines. The slave machines perform lookups for the respective label and global rank, streaming results to the coordinating machine, which in turn streams the final results to the initiating agent.

1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485

Collating the raw data for the focus view is more expensive than a simple lookup on one targeted machine—although helped by the hash-placement strategy, potentially many lookups may be required. We mitigate the expense using some application-level LRU caching, where the coordinating machine caches not only keyword snippet results and focus view results but also the labels and ranks for predicates and objects: in particular, this would save repetitive lookups on commonly encountered properties and classes.

1486  
1487  
1488  
1489  
1490  
1491  
1492

### 14.11.3 Related Work

1493

Besides query-processing components for systems referenced in Sect. 14.10.3, other types of query processing have been defined in the literature which do not rely on data warehousing approaches. 1494  
1495  
1496

The system presented in [61] leverages Linked Data principles to perform live lookups on Linked Data sources, rendering and displaying resulting data; however, such an approach suffers from low recall and inability to independently service keyword queries. In [57], we have described an approach which uses a lightweight hashing-based index structure—viz. a Q-Tree—for mapping structured queries to Linked Data sources which could possibly provide pertinent information; these sources are then retrieved and query processing performed. Such approaches suffer from poorer recall than data-warehousing approaches but enable the provision of up-to-date results to users, which is particularly expedient for query processing over highly dynamic sources. We could investigate inclusion of a live-lookup component in SWSE for a subset of queries which we identify to be best answered by means of live lookup; however, further research in this area is required to identify such queries and to investigate the performance of such a system. 1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509

Preliminary work on the DARQ [113] system provides federated query processing over a set of autonomous independent SPARQL endpoints. Such an approach may allow for increased query recall; however, the performance of such a system is still an open question; also, keyword search is still not a standard SPARQL operation, and thus, federating keyword queries would probably require manual wrappers for different SPARQL endpoints. 1510  
1511  
1512  
1513  
1514  
1515

### 14.11.4 Future Directions and Open Research Question

1516

With respect to current query-processing capabilities, our underlying index structures have proven scalable. However, the focus - view currently requires on average hundreds—but possibly tens or hundreds of thousands—of lookups for labels and ranks. Given that individual result sizes are likely to continue to grow, we will need to incorporate one of the following optimizations: (1) we can build a specialized join index which precomputes and stores focus-view results, requiring one atomic lookup for the entire focus-view result at the cost of longer indexing time, and (judging from our evaluation) a doubling of structured-index size, and/or (2) we can generate a top- $k$  focus-view result, paginating the view of a single entity and only retrieving incremental segments of the view—possibly asynchronously. 1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526

Extending the query processing to handle more complex queries is a topic of importance when considering extension and improvement of the current spartan UI. In order to fully realize the potential benefits of querying over structured data, we need to be able to perform optimized query processing. For querying data, there is 1527  
1528  
1529  
1530

a trade-off between the scalability of the approach and the expressivity of the query 1531  
language used. 1532

In the general case, joins are expensive operations, and when attempting to 1533  
perform arbitrary joins on very large datasets, the system either consumes a large 1534  
amount of resources per query or becomes slow. Some systems (such as [82]) solve 1535  
the scalability issue by partitioning the datasets into smaller units and have the user 1536  
select a subdataset before further browsing or querying; however, such a solution 1537  
impinges on the data-integration properties of RDF which provides the *raison d'être* 1538  
of a system such as SWSE. Another solution is to precompute joins, allowing for 1539  
direct lookup of results emulating the current approach; however, materializing 1540  
joins can lead to quadratic growth in index sizes. Investigation of partial join 1541  
materialization—perhaps based on the expense of a join operation, materialized size 1542  
of join, runtime caching, etc.—may enable sufficiently optimized query processing. 1543

Another open research question here is how to optimize for top- $k$  querying in 1544  
queries involving joins; joins at large scale can potentially lead to the access of 1545  
large - volumes of intermediary data, used to compute a final small result size; thus, 1546  
the question is how top- $k$  query processing can be used to immediately retrieve the 1547  
best results for joins, allowing, for example, path queries in the UI (joins on objects 1548  
and subject) such that large intermediate data volumes need not be accessed, and 1549  
rather than the approach of joining several attribute restrictions (e.g., facets) as done 1550  
in the threshold algorithm [44]. 1551

## 14.12 User Interface

1552

Having discussed data acquisition, enhancing, analysis, indexing, and query- 1553  
processing components, we have now come full circle and arrived at the user-facing 1554  
component. The user interface offers two basic operations: keyword search, where 1555  
the user specifies a set of keywords and the system returns with a list of matching 1556  
entities, and object focus, where the user can navigate to a consolidated view of all 1557  
information available for one entity (see Sect. 14.2). Our user interface uses XSLT 1558  
to convert the raw data returned by the query processor into result pages, offering 1559  
a declarative means of specifying user interface rendering and ultimately providing 1560  
greater flexibility for tweaking presentation. 1561

### 14.12.1 Related Work

1562

There has been considerable work on rendering and displaying RDF data; such 1563  
systems include BrowseRDF [105], Explorator [4], gFacet [66], Haystack [84], 1564

Longwell,<sup>33</sup> Piggybank [81], (Power)Magpie [51], Marbles,<sup>34</sup> RKBExplorer,<sup>35</sup> Tabulator [10], Zitgist,<sup>36</sup> as well as user interfaces for previously mentioned engines such as Falcons, Sig.ma, Sindice, Swoogle, Watson, etc. 1565  
1566  
1567

Fresnel [110] has defined an interesting approach to overcome the difficulty of displaying RDF in a domain-agnostic way by providing a vocabulary for describing how RDF should be rendered, thus allowing for the declarative provision of schema-specific views over data; some user interfaces have been proposed to exploit Fresnel, including LENA [87]; however, Fresnel has not seen widespread adoption on the Web thus far. 1568  
1569  
1570  
1571  
1572  
1573

### 14.12.2 Future Directions and Open Research Questions

1574

First, we must review the performance of the UI with respect to generating result pages—we had not previously considered this issue, but under high loads, UI result generation seems to be a significant factor in deciding response times. 1575  
1576  
1577

With respect to functionality, we currently do not fully exploit the potential offered by richly structured data. First, such data could power a large variety of visualizations: for example, to render SIMILE’s timeline view<sup>37</sup> or a GoogleMap view.<sup>38</sup> Countless other visualizations are possible for a history and examples of visualizations, cf. [48]. Research into rendering and visualizing large graph-structured datasets—particularly user evaluation thereof—could lead to novel user interfaces which better suit and exploit such information. 1578  
1579  
1580  
1581  
1582  
1583  
1584

Second, offering only keyword search and entity browsing removes the possibility of servicing more expressive queries which offer users more direct answers; however, designing a system for domain-agnostic users to formulate such queries in an intuitive manner—and one which is guided by the underlying data to avoid empty results where possible—has proven nontrivial. We have made first experiments with more expressive user interfaces for interacting with data through the VisiNav system<sup>39</sup> [55], which supports faceted browsing [129], path traversal [5], and data visualizations on top of the keyword search and focus operations supported by SWSE. Within VisiNav, we encourage users to incrementally create expressive queries while browsing, as opposed to having a formal query formulation step—users are offered navigation choices which lead to nonempty results. However, for such extra functionality—specifically the cost of querying associated with arbitrary 1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596

<sup>33</sup><http://simile.mit.edu/wiki/Longwell>

<sup>34</sup><http://marbles.sourceforge.net/>

<sup>35</sup><http://www.rkbexplorer.com/explorer/>

<sup>36</sup><http://dataviewer.zitgist.com/>

<sup>37</sup><http://www.simile-widgets.org/timeline/>

<sup>38</sup><http://maps.google.com/>

<sup>39</sup><http://visinav.deri.org/>

join paths—VisiNav must make scalability trade-offs: VisiNav can currently handle 1597  
in the realm of tens of millions of statements. 1598

Efforts to provide guided construction of structured queries (e.g., cf. [100]) may 1599  
be useful to so-called power-users; however, such methods again rely on some 1600  
knowledge of the schema of the pertinent data and query. Other efforts to match 1601  
keyword searches to structured queries (e.g., cf. [28, 94, 119]) could bring together 1602  
the ease of use of Web search engines and the precise answers of structured data 1603  
querying; however, again such formulations still require some knowledge of the 1604  
schema(ta) of the data, and which types of entities link to which by what type of link. 1605

For the moment, we focus on providing basic functionality as should be familiar 1606  
to many Web users. Previous incarnations of SWSE offered more complex user- 1607  
interaction models, allowing, e.g., filtering of results based on type, traversing 1608  
inlinks for an entity, and traversing links from a collection of entities. From informal 1609  
feedback received, we realized that features such as inlink traversal (and the notion 1610  
of directionality) were deemed confusing by certain users—or at least by our 1611  
implementation thereof.<sup>40</sup> We are thus more cautious about implementing additional 1612  
features in the user interface, aiming for minimalistic display and interaction. One 1613  
possible solution is to offer different versions of the user interface, for example, 1614  
a default system offering simple keyword search for casual users and an optional 1615  
system offering more complex functionality for power users. In such regards, user 1616  
evaluation (currently out of scope) would be of utmost importance in making such 1617  
design choices. 1618

We also wish to investigate the feasibility of offering programmatic interfaces 1619  
through SWSE.<sup>41</sup> The main requirements for such APIs are performance and 1620  
reliability: the API has to return results fast enough to enable interactive applications 1621  
and has to have high uptime to encourage adoption by external services. Full 1622  
SPARQL is likely too powerful (and hence too expensive to evaluate) to provide 1623  
stable, complete, and fast responses for. One possible workaround is to provide 1624  
time-out queries, which return as many answers as can be serviced in a fixed time 1625  
period; another possible solution is to offer top- $k$  query processing, or a well- 1626  
supported subset of SPARQL (e.g., DESCRIBE queries and conjunctive queries 1627  
with a limited number of joins or containing highly selective patterns) such that 1628  
could serve as a foundation for visualizations and other applications leveraging the 1629  
integrated Web data in SWSE. 1630

<sup>40</sup>In any case, our reasoning engine supports the `owl:inverseOf` construct which solves the problem of directionality, and we would hope that most (object) properties define a corresponding inverse property.

<sup>41</sup>Please note that practical limitations with respect to the availability and administration of physical machines have restricted our ability to provide such interfaces with high reliability; indeed, we used to offer time-out SPARQL queries over  $\sim 1.5$  bn statements through YARS2, but for the meantime, we can no longer support such a service.

## 14.13 Conclusion

1631

In this chapter, we have presented the results of research carried out as part of the SWSE project over several years. In particular, we have described how we adapted the architecture of large-scale Web search engines to the case of structured data. We have presented lightweight algorithms which demonstrate the data-integration possibilities for Linked Data and shown how such algorithms can be made scalable using batch-processing techniques such as scans and sorts and how they can be deployed over a distributed architecture. We have argued for the importance of taking the source of information into account when handling arbitrary RDF Web data, showing how Linked Data principles can be leveraged for such purposes, particularly in our ranking and reasoning algorithms.

Research on how to integrate and interact with large amounts of data from a very diverse set of independent sources is fairly recent, as many characteristics of the research questions in the field became visible after the deployment of large amounts of data by a sizable body of data publishers. The traditional application development cycle for data-intensive applications is to model the data schema and build the application on top: data modeling and application development are tightly coupled. That process is separated on the semantic Web: data publishers just model and publish data, often with no particular application in mind. At the same time, the quality of data that a system such as SWSE operates over is perhaps as much of a factor in the system's utility as the design of the system itself.

Recently, there has been significant success with Linked Data where an active community publishes datasets in a broad range of topics and maintains and interlinks these datasets. Again, efforts such as DBpedia have lead to a much richer Web of Data than the one present when we began working on SWSE. However, data heterogeneity still poses problems—not so much for the underlying components of SWSE—but for the user-facing components and the users themselves: allowing domain-oblivious users to create flexible structured queries in a convenient and intuitive manner is still an open question. Indeed, the Web of Data still cannot compete with the vast coverage of the Web of Documents, and perhaps never will [111].

That said, making Web data available for querying and navigation has significant scientific and commercial potential. First, the Web becomes subject to scientific analysis [12]: understanding the implicit connections and structure of the Web of Data can help to reveal new understandings of collaboration patterns and the processes by which networks form and evolve. Second, aggregating and enhancing scientific data published on the Web can help scientists to more easily perform data-intensive research, in particular allowing for the arbitrary repurposing of published datasets which can subsequently be used in ways unforeseen by the original publisher. Third, making the Web of Data available for interactive querying, browsing, and navigation has applications in areas such as e-commerce and e-health, allowing data analysts in such fields to pose complex structured queries over a dataset aggregated from multitudinous relevant sources.

**Acknowledgments** The work presented herein was funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2) and by an IRCSET postgraduate scholarship. 1673  
1674

## References

1675

1. Alani, H., Brewster, C., Shadbolt, N.: Ranking ontologies with AKTiveRank. 5th international semantic web conference, pp. 1–15 (2006) 1676  
1677
2. Alani, H., Dasmahapatra, S., O'Hara, K., Shadbolt, N.: Identifying communities of practice through ontology network analysis. IEEE Intel. Syst. **18**(2), 18–25 (2003) 1678  
1679
3. Anyanwu, K., Maduko, A., Sheth, A.: SemRank: ranking complex relationship search results on the semantic web. 14th International Conference on World Wide Web, pp. 117–127 (2005). DOI <http://doi.acm.org/10.1145/1060745.1060766> 1680  
1681  
1682
4. de Araújo, S.F.C., Schwabe, D.: Explorator: a tool for exploring RDF data through direct manipulation. Linked data on the web WWW2009 workshop (LDOW2009) (2009) 1683  
1684
5. Athanasis, N., Christophides, V., Kotzinos, D.: Generating On the fly queries for the semantic web: the ICS-FORTH graphical RQL interface (GRQL). 3rd international semantic web conference, pp. 486–501 (2004) 1685  
1686  
1687
6. Balmin, A., Hristidis, V., Papakonstantinou, Y.: Objectrank: authority-based keyword search in databases. Proceedings of the 13th International Conference on very Large Data Bases, pp. 564–575 (2004) 1688  
1689  
1690
7. Batsakis, S., Petrakis, E.G.M., Milios, E.: Improving the performance of focused web crawlers. Data Knowledge Eng. **68**(10), 1001–1013 (2009). DOI <http://dx.doi.org/10.1016/j.datak.2009.04.002> 1691  
1692  
1693
8. Bechhofer, S., Volz, R.: Patching Syntax in OWL Ontologies. International semantic web conference (ISWC 2004), Lecture Notes in Computer Science, vol. 3298, pp. 668–682. Springer, Berlin, Heidelberg, New York (2004) 1694  
1695  
1696
9. Berners-Lee, T.: Linked Data. Design issues for the World Wide Web, World Wide Web Consortium (2006). <http://www.w3.org/DesignIssues/LinkedData.html> 1697  
1698
10. Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Lerer, A., Sheets, D.: Tabulator: exploring and analyzing linked data on the semantic web. In Proceedings of the 3rd International Semantic Web user Interaction Workshop (2006) 1699  
1700  
1701
11. Berners-Lee, T., Fielding, R., Masinter, L.: Uniform resource identifier (URI): generic syntax. RFC 3986 (2005). <http://tools.ietf.org/html/rfc3986> 1702  
1703
12. Berners-Lee, T., Hall, W., Hendler, J., Shadbolt, N., Weitzner, D.J.: Creating a science of the web. Science **313**(11) (2006) 1704  
1705
13. Bizer, C., Cyganiak, R.: D2R server – publishing relational databases on the web as SPARQL Endpoints. ISWC (2006). (poster) 1706  
1707
14. Bizer, C., Heath, T., Berners-Lee, T.: Linked data – the story so far. Int. J. Semant. Web Inf. Syst. **5**(3), 1–22 (2009) 1708  
1709
15. Boldi, P., Codenotti, B., Santini, M., Vigna, S., Vigna, S.: UbiCrawler: a scalable fully distributed web crawler. Soft. Pract. Exp. **34**, 2004 1710  
1711
16. Bonatti, P.A., Hogan, A., Polleres, A., Sauro, L.: Robust and Scalable Linked Data Reasoning Incorporating Provenance and Trust Annotations. J. Web Semant. (2011, in Press) 1712  
1713
17. Bouquet, P., Stoermer, H., Mancioppi, M., Giacomuzzi, D.: OkkaM: towards a solution to the “Identity Crisis” on the semantic web. Proceedings of SWAP 2006, the 3rd Italian Semantic Web Workshop, CEUR Workshop Proceedings, vol. 201 (2006) 1714  
1715  
1716
18. Brewer, E.A.: Combining Systems and Databases: A Search Engine Retrospective, pp. 711–724. MIT Press, Cambridge, MA (2005) 1717  
1718
19. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Comput. Networks **30**(1–7), 107–117 (1998) 1719  
1720

20. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: a generic architecture for storing and querying RDF and RDF schema. 2nd international semantic web conference, pp. 54–68. Springer, Berlin, Heidelberg, New York (2002) 1721  
1722  
1723
21. Cai, D., He, X., Wen, J., Ma, W.: Block-level link analysis. 27th international ACM SIGIR conference on research and development in information retrieval, pp. 440–447 (2004) 1724  
1725
22. Caverlee, J., Liu, L.: QA-Pagelet: data preparation techniques for large-scale data analysis of the deep web. *IEEE Trans. Knowl. Data Eng.* **17**(9), 1247–1262 (2005) 1726  
1727
23. Chakrabarti, S., van den Berg, M., Dom, B.: Focused crawling: a new approach to topic-specific web resource discovery. *Comput. Networks* **31**(11–16), 1623–1640 (1999) 1728  
1729
24. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.: Bigtable: a distributed storage system for structured data. *OSDI*, pp. 205–218 (2006) 1730  
1731  
1732
25. Chang, K.C.C., He, B., Zhang, Z.: Toward large scale integration: building a MetaQuerier over databases on the web. *CIDR*, pp. 44–55 (2005) 1733  
1734
26. Chen, Z., Kalashnikov, D.V., Mehrotra, S.: Exploiting relationships for object consolidation. *IQIS '05: Proceedings of the 2nd International Workshop on Information Quality in Information Systems*, pp. 47–58. ACM, New York, NY, USA (2005). DOI <http://doi.acm.org/10.1145/1077501.1077512> 1735  
1736  
1737  
1738
27. Cheng, G., Ge, W., Wu, H., Qu, Y.: Searching semantic web objects based on class hierarchies. *Proceedings of Linked Data on the Web Workshop* (2008) 1739  
1740
28. Cheng, G., Qu, Y.: Searching linked objects with falcons: approach, implementation and evaluation. *Int. J. Semant. Web Inf. Syst.* **5**(3), 49–70 (2009) 1741  
1742
29. Cheng, T., Chang, K.C.C.: Entity search engine: towards Agile best-effort information integration over the web. *CIDR*, pp. 108–113 (2007) 1743  
1744
30. d'Aquin, M., Sabou, M., Motta, E., Angeletou, S., Gridinoc, L., Lopez, V., Zablith, F.: What can be done with the semantic web? an overview Watson-based applications. *SWAP* (2008) 1745  
1746
31. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. *OSDI*, pp. 137–150 (2004) 1747  
1748
32. Decker, S., Erdmann, M., Fensel, D., Studer, R.: Ontobroker: ontology based access to distributed and semi-structured information. *DS-8: IFIP TC2/WG2.6 eighth working conference on database semantics*, pp. 351–369. Kluwer, B.V., Deventer, The Netherlands, The Netherlands (1998) 1749  
1750  
1751  
1752
33. Delbru, R., Polleres, A., Tummarello, G., Decker, S.: Context dependent reasoning for semantic documents in Sindice. *Proceedings of the 4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2008)*. Karlsruhe, Germany (2008). URL <http://www.polleres.net/publications/delbru-et-al-2008.pdf> 1753  
1754  
1755  
1756
34. Delbru, R., Toupikov, N., Catasta, M., Tummarello, G.: A node indexing scheme for web entity retrieval. *Proceedings of the Extended Semantic Web Conference (ESWC 2010)* (2010) 1757  
1758  
1759
35. Delbru, R., Toupikov, N., Catasta, M., Tummarello, G., Decker, S.: Hierarchical link analysis for ranking web data. *Proceedings of the Extended Semantic Web Conference (ESWC 2010)* (2010) 1760  
1761  
1762
36. Dietze, H., Schroeder, M.: Semplore: a scalable IR approach to search the web of data. *BMC Bioinfor.* **10** (2009) 1763  
1764
37. Diligenti, M., Coetzee, F., Lawrence, S., Giles, C.L., Gori, M.: Focused crawling using context graphs. *VLDB '00: Proceedings of the 26th International Conference on very Large Data Bases*, pp. 527–534. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000) 1765  
1766  
1767
38. Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R.S., Peng, Y., Reddivari, P., Doshi, V.C., Sachs, J.: Swoogle: a search and metadata engine for the semantic web. *13th ACM conference on information and knowledge management*. ACM, New York (2004) 1768  
1769  
1770
39. Ding, L., Pan, R., Finin, T., Joshi, A., Peng, Y., Kolari, P.: Finding and ranking knowledge on the semantic web. *4th international semantic web conference*, pp. 156–170 (2005) 1771  
1772

40. Dong, H., Hussain, F.K., Chang, E.: State of the art in semantic focused crawlers. ICCSA '09: Proceedings of the International Conference on Computational Science and its Applications, pp. 910–924. Springer, Berlin, Heidelberg (2009). DOI [http://dx.doi.org/10.1007/978-3-642-02457-3\\_74](http://dx.doi.org/10.1007/978-3-642-02457-3_74) 1773  
1774  
1775  
1776
41. Ehrig, M., Maedche, A.: Ontology-focused crawling of Web documents. SAC '03: Proceedings of the 2003 ACM Symposium on Applied Computing, pp. 1174–1178. ACM, New York, NY, USA (2003). DOI <http://doi.acm.org/10.1145/952532.952761> 1777  
1778  
1779
42. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2007) 1780  
1781
43. Erling, O., Mikhailov, I.: RDF support in the Virtuoso DBMS. CSSW, pp. 59–68 (2007) 1782
44. Fagin, R.: Combining fuzzy information from multiple systems (extended abstract). PODS '96: Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 216–226. ACM, New York (1996). DOI <http://doi.acm.org/10.1145/237661.237715> 1783  
1784  
1785  
1786
45. Fensel, D., van Harmelen, F.: Unifying reasoning and search to web scale. IEEE Inter. Comput. **11**(2), 94–96 (2007). DOI <http://doi.ieeecomputersociety.org/10.1109/MIC.2007.51> 1787  
1788
46. Fielding, R., Gettys, J., Mogul, J., Nielsen, H.F., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext transfer protocol – HTTP/1.1. RFC 2616 (1999). <ftp://ftp.isi.edu/in-notes/rfc2616.txt> 1789  
1790  
1791
47. Franz, T., Schultz, A., Sizov, S., Staab, S.: TripleRank: ranking semantic web data by tensor decomposition. 8th international semantic web conference (ISWC2009) (2009) 1792  
1793
48. Friendly, M.: A brief history of data visualization. In: Chen, C., Härdle, W., Unwin, A. (eds.) Handbook of Computational Statistics: Data Visualization, vol. III. Springer, Heidelberg (2006) 1794  
1795  
1796
49. Glaser, H., Millard, I., Jaffri, A.: RKBExplorer.com: a knowledge driven infrastructure for linked data providers. ESWC Demo, Lecture Notes in Computer Science, pp. 797–801. Springer, Berlin, Heidelberg, New York (2008) 1797  
1798  
1799
50. Grau, B.C., Motik, B., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology language: profiles. W3C Working Draft (2008). <http://www.w3.org/TR/owl2-profiles/> 1800  
1801
51. Gridinoc, L., Sabou, M., d'Aquin, M., Dzbor, M., Motta, E.: Semantic browsing with PowerMagpie. ESWC, pp. 802–806 (2008) 1802  
1803
52. Guha, R.V., McCool, R., Fikes, R.: Contexts for the Semantic Web. 3rd International Semantic Web Conference, Hiroshima (2004) 1804  
1805
53. Halpin, H., Hayes, P.J., McCusker, J.P., McGuinness, D.L., Thompson, H.S.: When owl:sameAs isn't the same: an analysis of identity in linked data. International Semantic Web Conference (1), pp. 305–320 (2010) 1806  
1807  
1808
54. Harris, S., Lamb, N., Shadbolt, N.: 4store: The Design and Implementation of a Clustered RDF Store. 5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009) (2009) 1809  
1810  
1811
55. Harth, A.: Visinav: a system for visual search and navigation on web data. J. Web Semat. **8**(4), 348–354 (2010) 1812  
1813
56. Harth, A., Decker, S.: Optimized index structures for querying RDF from the web. 3rd Latin American Web Congress, pp. 71–80. IEEE Press, New York (2005) 1814  
1815
57. Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K.U., Umbrich, J.: Data summaries for on-demand queries over linked data. WWW, pp. 411–420 (2010) 1816  
1817
58. Harth, A., Kinsella, S.: Topdis: tensor-based ranking for data search and navigation. Technical Report, DERI (2009) 1818  
1819
59. Harth, A., Kinsella, S., Decker, S.: Using naming authority to rank data and ontologies for web search. 8th International Semantic Web Conference (ISWC 2009) (2009) 1820  
1821
60. Harth, A., Umbrich, J., Hogan, A., Decker, S.: YARS2: a federated repository for querying graph structured data from the web. 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, pp. 211–224 (2007) 1822  
1823  
1824
61. Hartig, O., Bizer, C., Freytag, J.C.: Executing SPARQL queries over the web of linked data. International Semantic Web Conference, pp. 293–309 (2009) 1825  
1826

62. Hatcher, E., Gospodnetic, O.: Lucene in Action. Manning Publications (2004) 1827
63. Hayes, P.: RDF semantics. W3C Recommendation (2004). <http://www.w3.org/TR/rdf-mt/> 1828
64. He, B., Patel, M., Zhang, Z., Chang, K.C.C.: Accessing the deep web. Commun. ACM **50**(5), 1829  
94–101 (2007) 1830
65. Heflin, J., Hendler, J., Luke, S.: SHOE: a knowledge representation language for internet 1831  
applications. Technical Report CS-TR-4078, Department of Computer Science, University of 1832  
Maryland (1999) 1833
66. Heim, P., Ziegler, J., Lohmann, S.: gFacet: a browser for the web of data. Proceedings of the 1834  
International Workshop on Interacting with Multimedia Content in the Social Semantic Web 1835  
(IMC-SSW'08), pp. 49–58. CEUR-WS (2008) 1836
67. Heydon, A., Najork, M.: Mercator: a scalable, extensible web crawler. World Wide Web **2**, 1837  
219–229 (1999) 1838
68. Hirai, J., Raghavan, S., Garcia-Molina, H., Paepcke, A.: WebBase: a repository of Web pages'. 1839  
Comput. Networks **33**(1–6), 277–293 (2000) 1840
69. Hitzler, P., van Harmelen, F.: A reasonable semantic web. Semant. Web Interop. Usabil. Appl. 1841  
**1** (2010) 1842
70. Hogan, A.: Exploiting RDFS and OWL for integrating heterogeneous, large-scale, linked data 1843  
corpora. Ph.D. thesis, Digital Enterprise Research Institute, National University of Ireland, 1844  
Galway (2011). Available from <http://aidanhogan.com/docs/thesis/> 1845
71. Hogan, A., Decker, S.: On the ostensibly silent 'W' in OWL 2 RL. Third International 1846  
Conference on Web Reasoning and Rule Systems, (RR2009), pp. 118–134 (2009) 1847
72. Hogan, A., Harth, A., Decker, S.: ReConRank: a scalable ranking method for semantic 1848  
web data with context. 2nd Workshop on Scalable Semantic Web Knowledge Base Systems 1849  
(SSWS2006) (2006) 1850
73. Hogan, A., Harth, A., Decker, S.: Performing object consolidation on the semantic web data 1851  
graph. 1st I3 Workshop: Identity, Identifiers, Identification Workshop (2007) 1852
74. Hogan, A., Harth, A., Passant, A., Decker, S., Polleres, A.: Weaving the pedantic web. Linked 1853  
Data on the Web WWW2010 Workshop (LDOW2010) (2010) 1854
75. Hogan, A., Harth, A., Polleres, A.: Scalable authoritative OWL reasoning for the web. Int. J. 1855  
Semant. Web Inf. Syst. **5**(2) (2009) 1856
76. Hogan, A., Harth, A., Umbrich, J., Kinsella, S., Polleres, A., Decker, S.: Searching and 1857  
browsing Linked Data with SWSE: the semantic web search engine. J. Web Semant. (2011). 1858  
DOI DOI:10.1016/j.websem.2011.06.004 1859
77. Hogan, A., Pan, J.Z., Polleres, A., Decker, S.: SAOR: template rule optimisations for dis- 1860  
tributed reasoning over 1 billion linked data triples. International Semantic Web Conference 1861  
(2010) 1862
78. Hogan, A., Polleres, A., Umbrich, J., Zimmermann, A.: Some entities are more equal than 1863  
others: statistical methods to consolidate Linked Data. 4th International Workshop on New 1864  
Forms of Reasoning for the Semantic Web: Scalable and Dynamic (NeFoRS2010) (2010) 1865
79. ter Horst, H.J.: Completeness, decidability and complexity of entailment for RDF Schema 1866  
and a semantic extension involving the OWL vocabulary. J. Web Semant. **3**, 79–115 (2005) 1867
80. Hu, W., Chen, J., Qu, Y.: A self-training approach for resolving object coreference on the 1868  
Semantic Web. WWW, pp. 87–96 (2011) 1869
81. Huynh, D., Mazzocchi, S., Karger, D.R.: Piggy bank: experience the semantic web inside 1870  
your web browser. J. Web Semat. **5**(1), 16–27 (2007) 1871
82. Huynh, D.F., Karger, D.: Parallax and companion: set-based browsing for the data web. Avail- 1872  
able online (2008-12-15) <http://davidhuynh.net/media/papers/2009/www2009-parallax.pdf> 1873
83. Jiang, X.M., Xue, G.R., Song, W.G., Zeng, H.J., Chen, Z., Ma, W.Y.: Exploiting PageRank at 1874  
different block level . 5th International Conference on Web Information Systems, pp. 241–252 1875  
(2004) 1876
84. Karger, D.R., Bakshi, K., Huynh, D., Quan, D., Sinha, V.: Haystack: a general-purpose 1877  
information management tool for end users based on semistructured data. CIDR, pp. 13–26 1878  
(2005) 1879

85. Kiryakov, A., Ognyanoff, D., Velkov, R., Tashev, Z., Peikov, I.: LDSR: a reason-able view to the web of linked data. Semantic Web Challenge (ISWC2009) (2009) 1880  
1881
86. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. *J. ACM* **46**(5), 604–632 1882  
1883 (1999)
87. Koch, J., Franz, T.: LENA – browsing RDF data more complex than Foaf. International Semantic Web Conference (Posters & Demos) (2008) 1884  
1885
88. Kolovski, V., Wu, Z., Eadon, G.: Optimizing enterprise-scale OWL 2 RL reasoning in a relational database system. International Semantic Web Conference (2010) 1886  
1887
89. Kotoulas, S., Oren, E., van Harmelen, F.: Mind the data skew: distributed inferencing by speeddating in elastic regions. *WWW*, pp. 531–540 (2010) 1888  
1889
90. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. *Operat. Syst. Rev.* **44**(2), 35–40 (2010) 1890  
1891
91. Lee, H.T., Leonard, D., Wang, X., Loguinov, D.: IRLbot: scaling to 6 billion pages and beyond. *ACM Trans. Web* **3**(3), 1–34 (2009). DOI <http://doi.acm.org/10.1145/1541822.1541823> 1892  
1893  
1894
92. Lei, Y., Uren, V., Motta, E.: Semsearch: a search engine for the semantic web. 14th International Conference on Knowledge Engineering and Knowledge Management, pp. 238–245 (2006) 1895  
1896  
1897
93. Liu, B., Hu, B.: HPRD: a high performance RDF database. *NPC*, pp. 364–374 (2007) 1898
94. Lopez, V., Uren, V.S., Motta, E., Pasin, M.: AquaLog: an ontology-driven question answering system for organizational semantic intranets. *J. Web Semat.* **5**(2), 72–105 (2007) 1899  
1900
95. Manola, F., Miller, E., McBride, B.: RDF Primer. W3C Recommendation (2004). <http://www.w3.org/TR/rdf-primer/> 1901  
1902
96. Meditskos, G., Bassiliades, N.: DLEJena: a practical forward-chaining OWL 2 RL reasoner combining Jena and Pellet. *J. Web Semat.* **8**(1), 89–94 (2010) 1903  
1904
97. Melnik, S., Raghavan, S., Yang, B., Garcia-Molina, H.: Building a distributed full-text index for the web. 10th International World Wide Web Conference, Hong Kong, pp. 396–406 (2001) 1905  
1906  
1907
98. Michalowski, M., Thakkar, S., Knoblock, C.A.: Exploiting secondary sources for automatic object consolidation. Proceeding of 2003 KDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation (2003) 1908  
1909  
1910
99. Minack, E., Siberski, W., Nejdl, W.: Benchmarking fulltext search performance of RDF stores. *ESWC*, pp. 81–95 (2009) 1911  
1912
100. Möller, K., Ambrus, O., Josan, L., Handschuh, S.: A visual interface for building SPARQL queries in Konduit. International Semantic Web Conference (Posters & Demos) (2008) 1913  
1914
101. Najork, M., Zaragoza, H., Taylor, M.: HITS on the web: how does it compare? Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, p. 478. ACM, New York (2007) 1915  
1916  
1917
102. Neumann, T., Weikum, G.: The RDF-3X engine for scalable management of RDF data. *VLDDB J.* **19**(1), 91–113 (2010) 1918  
1919
103. Newcombe, H.B., Kennedy, J.M., Axford, S.J., James, A.P.: Automatic linkage of vital records: computers can be used to extract "follow-up" statistics of families from files of routine records. *Science* **130**, 954–959 (1959) 1920  
1921  
1922
104. Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., Tummarello, G.: Sindice.com: a document-oriented lookup index for open linked data. *Int. J. Metadata Semant. Ontol.* **3**(1), 37–52 (2008) 1923  
1924  
1925
105. Oren, E., Delbru, R., Decker, S.: Extending faceted navigation for RDF data. International Semantic Web Conference, pp. 559–572 (2006) 1926  
1927
106. Oren, E., Kotoulas, S., Anadiotis, G., Siebes, R., ten Teije, A., van Harmelen, F.: Marvin: distributed reasoning over large-scale Semantic Web data. *J. Web Semat.* **7**(4), 305–316 1928  
1929  
1930 (2009)
107. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: bringing order to the web. Technical report, Stanford Digital Library Technologies Project (1998) 1931  
1932

108. Pant, G., Srinivasan, P.: Learning to crawl: comparing classification schemes. *ACM Trans. Inf. Syst.* **23**(4), 430–462 (2005) 1933  
1934
109. Pérez-Agüera, J.R., Arroyo, J., Greenberg, J., Iglesias, J.P., Fresno, V.: Using BM25F for semantic search. 3rd International Semantic Search Workshop (SEMSEARCH) (2010) 1935  
1936
110. Pietriga, E., Bizer, C., Karger, D.R., Lee, R.: Fresnel: a browser-independent presentation vocabulary for RDF. International Semantic Web Conference, pp. 158–171 (2006) 1937  
1938
111. Polleres, A., Hogan, A., Harth, A., Decker, S.: Can we ever catch up with the Web? *Semant. Web Interoper. Usabil. Appl.* **1** (2010) 1939  
1940
112. Prud'hommeaux, E., (eds.), Seaborne, A.: SPARQL query language for RDF. W3C Recommendation (2008). <http://www.w3.org/TR/rdf-sparql-query/> 1941  
1942
113. Quilizzi, B., Leser, U.: Querying Distributed RDF Data Sources with SPARQL. ESWC, pp. 524–538 (2008) 1943  
1944
114. Raghavan, S., Garcia-Molina, H.: Crawling the hidden web. VLDB, pp. 129–138 (2001) 1945
115. Sabou, M., Baldassarre, C., Gridinoc, L., Angeletou, S., Motta, E., d'Aquin, M., Dzbor, M.: WATSON: a gateway for the semantic web. ESWC 2007 poster session (2007-06) 1946  
1947
116. Smith, M.K., Welty, C., McGuinness, D.L.: OWL web ontology language guide. W3C Recommendation (2004). <http://www.w3.org/TR/owl-guide/> 1948  
1949
117. Stonebraker, M.: The case for shared nothing. *IEEE Database Eng. Bull.* **9**(1), 4–9 (1986) 1950
118. Thelwall, M., Stuart, D.: Web crawling ethics revisited: cost, privacy, and denial of service. *J. Am. Soc. Inform. Sci. Technol.* **57**, 1771–1779 (2006) 1951  
1952
119. Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering, pp. 405–416 (2009). DOI <http://dx.doi.org/10.1109/ICDE.2009.119> 1953  
1954
120. Tummarello, G., Cyganiak, R., Catasta, M., Danielczyk, S., Decker, S.: Sig.ma: live views on the web of data. Semantic Web Challenge (2009) 1955  
1956
121. Umbrich, J., Harth, A., Hogan, A., Decker, S.: Four heuristics to guide structured content crawling. Proceedings of the 2008 Eighth international conference on web engineering- Volume 00, pp. 196–202. IEEE Computer Society, Silver Spring, MD (2008) 1959  
1960  
1961
122. Urbani, J., Kotoulas, S., Maassen, J., van Harmelen, F., Bal, H.E.: OWL reasoning with WebPIE: calculating the closure of 100 Billion Triples. ESWC, vol. 1, pp. 213–227 (2010) 1962  
1963
123. Urbani, J., Kotoulas, S., Oren, E., van Harmelen, F.: Scalable distributed reasoning using MapReduce. International Semantic Web Conference (ISWC 2009), vol. 5823, pp. 634–649. Springer, Washington DC, USA (2009) 1964  
1965  
1966
124. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and maintaining links on the web of data. International Semantic Web Conference, pp. 650–665 (2009) 1967  
1968
125. Wang, T.D., Parsia, B., Hendler, J.A.: A survey of the web ontology landscape. International Semantic Web Conference, pp. 682–694 (2006) 1969  
1970
126. Weaver, J., Hendler, J.A.: Parallel materialization of the finite RDFS closure for hundreds of millions of triples. International Semantic Web Conference (ISWC2009), pp. 682–697 (2009) 1971  
1972
127. Wei, W., Barnaghi, P.M., Bargiela, A.: Search with meanings: an overview of semantic search systems. *Int. J. Commun. SIWN* **3**, 76–82 (2008) 1973  
1974
128. Weiss, C., Karras, P., Bernstein, A.: Hexastore: sextuple indexing for semantic web data management. PVLDB **1**(1), 1008–1019 (2008) 1975  
1976
129. Yee, K.P., Swearingen, K., Li, K., Hearst, M.: Faceted metadata for image search and browsing. SIGCHI Conference on Human factors in Computing Systems, pp. 401–408 (2003). DOI <http://doi.acm.org/10.1145/642611.642681> 1977  
1978

AUTHOR QUERIES

- AQ1. Please check if the edits made to the sentence “In the above version, rules...” are fine.
- AQ2. Kindly update Ref. [16].

UNCORRECTED PROOF

# Index

A( $k$ )index, 50  
Authoritative reasoning, 396  
  
Bisimulation, 50  
    forward and backward, 51  
    localized, 50  
Bitmap index, 36  
BTree, 34, 35  
  
Chase, 255, 260  
Chase failure, 275  
Class hierarchy, 43  
Cluster, 84  
Clustered index, 35  
Combined complexity, 260  
Conceptual dependencies, 263  
Conditional entity-centric query, 42, 43  
Conjunctive query, 259  
Connected component, 88  
Crawling, 371  
  
Data cleaning, 219  
Data complexity, 254, 260  
Data extraction, 348, 354  
Dataguide, 50  
Data integration, 353, 354, 356,  
    358  
Data quality, 219, 356  
Dataspace, 42  
Dereferencing, 369  
Description logics, 253  
Design pattern, 65  
DHTML, 175

Dictionary, 33  
Disk I/O, 37  
Distance-based graph index, 48  
DL-Lite, 254, 277  
  
Entity, 41  
    association, 43  
    attribute, 43  
    consolidation, 379  
    linkage, 223  
    relationship, 253  
    resolution, 192  
Exhaustivity, 80  
Exploratory search, 61  
  
Finite function generation, 274  
First-order rewritability, 272, 277  
Full-path, 83  
  
Graph, 82  
Graph database, 47  
GRIN index, 48  
gStore, 53  
  
Heterogeneous information, 226  
Hexastore, 35  
Hidden Markov model, 146  
HITS algorithm, 148  
Homomorphism, 258  
Horizontal representation, 37  
hRESTS, 282, 288  
Hungarian algorithm, 134

- 1-Index, 50  
2-Index, 50  
Inverted index, 42–45
- Jena, 38  
Join index, 40
- Key dependency, 259  
Keyword-based search, 129  
Keyword query, 42, 43, 110  
Keyword search, 168, 187  
KQB, 173  
KWilt, 180
- Labeled nulls, 258  
Linear building, 88  
Linkage method, 199  
Link discovery, 191  
Linked data, 3, 61, 192, 368  
Linked Data principles, 283, 291  
Linked Data Services, 282, 288  
Linked Open Data, 29  
Linked Web API, 283  
    functional coupling links, 283, 290, 294  
    functional similarity links, 283, 290, 293  
I-MASH platform, 298  
Linked Web API Explorer, 299, 300  
Linked Web API model, 289  
Linked Web API Repository, 299  
    selection patterns, 295  
LinQL, 195  
LinQuer, 192  
Literal indexing, 39
- MapReduce, 370  
Matching, 82  
Materialized view, 41  
Microdata, 3  
Microformats, 3  
MicroWSMO, 282, 288  
Monotonic building, 90  
Monotonicity, 92  
Munkres algorithm, 135
- Negative constraints, 256, 259, 276  
Non-conflicting, 266
- Object consolidation, 46, 379  
Ontology hijacking, 396
- Overlap, 80  
OWL-S, 282, 287
- Parameterizable index graph, 52  
Path, 83  
Payload, 49  
Permutations, 255  
 $P(k)$  index, 51  
PLD starvation, 375  
Probabilistic linkage database, 232  
Projection index, 35  
Property table, 38  
Prüfer sequences, 49
- Quadruple, 369  
Query construction graph, 113  
Query guide, 113  
Query processing, 402  
Query template, 110  
QUICK, 108
- Ranking, 80, 383  
RDF, 79  
RDFa, 3  
RDF-3X, 35  
Reasoner, 389  
Relational databases, 33  
Resource description framework, 367  
Resource identifier, 33  
    counter-based, 34  
    hash-based, 33  
RESTful services, 282, 286–289, 299. *See also* Web API
- SAWSDL, 287  
Scoring function, 85  
Search computing, 60, 72  
Search process, 64  
Semantic Link Discovery, 191  
Semantic query expansion, 83  
Semantic Web, 253  
Semantic Web Services, 282, 286  
Semantic wikis, 158  
Separability, 255, 265  
Set operations, 42  
Sindice, 44  
SOAP-based Web services, 282, 286  
Social web applications, 157  
Sound path, 84  
Source discovery, 337

- SPARQL query, 296  
Sparse index, 37  
Specificity, 80  
Strongly consistent, 276  
Structural index, 49, 408  
    approximate, 50  
    precise, 49  
Structural summary. *See* Structural index  
Structure index. *See* Structural index  
Structured data, 61  
Suffix array, 47  
System  $\Pi$ , 49
- $\tau$ -test, 92  
Text index, 399  
TF-IDF, 400  
Threshold, 92  
Threshold algorithm, 81  
 $T$ -index, 50  
Top-k, 80  
Tree labeling schemes, 47  
Triple store. *See* Vertical representation  
Tuple-generating dependency, 259  
Uncertain data, 356  
Uncertainty, 226  
Unclustered index, 34
- Vertical partitioning, 39  
Vertical representation, 34  
View-based query answering. *See* Materialized view  
Virtuoso, 36  
Viterbi algorithm, 130  
Volatile data, 226  
VS-tree, 53
- Web API, 282, 286  
Web mashup, 282, 288, 298  
    semantic mashup, 288  
Web of Data, 3  
Wrapper, 337–339, 341, 349, 353, 355, 356,  
    358  
Wrapper inference, 338  
WSMO, 282, 287  
WSMO-Lite, 282, 288